Kernel Methods and Introduction to Gaussian Processes

Prof. Nicholas Zabaras
University of Notre Dame
Notre Dame, IN, USA

Email: nzabaras@gmail.com
URL: https://www.zabaras.com/

November 13, 2017

Contents

Following closely: Bishop CM, *Pattern Recognition and Machine Learning*, Springer, 2006 (Chapter 6)

Training Data: Keep or Discard?

- We have considered linear parametric methods e.g. for regression.
- The form of the mapping $y(\boldsymbol{x}, \boldsymbol{w})$ from input $\boldsymbol{x}$ to output y is governed by $\boldsymbol{w}$.
- During the learning phase, the training data is used either
  - to obtain a point estimate of $\boldsymbol{w}$ or
  - posterior distribution $p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t})$.
- The training data is then discarded, and predictions for new inputs are based purely on the learned parameter vector $\boldsymbol{w}$.
- The same approach is used in nonlinear parametric models including neural networks.

Training Data: Keep or Discard?

- There are machine learning techniques, in which the training data points are kept and used for prediction.
  - Parzen probability density model: set of kernel functions centered on training data points
  - Nearest neighbors technique: assigning to each new test vector the same label as the closest example from the training set
- These are memory-based methods: storing the training set in order to make predictions for future data points.
- They require a metric to measure the similarity of any two vectors in input space.
- They are generally fast to 'train' but slow at making predictions for test data points.

Kernel Methods and Kernel Trick

- Many linear parametric models can be recast into an equivalent 'dual representation'
  - **Prediction is based on linear combinations of a kernel function evaluated at the training data points**
- For models which are based on a fixed nonlinear feature space mapping $\phi(\boldsymbol{x})$, the kernel function is given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$$

- Using the identity mapping for the feature space we obtain the linear kernel:

$$\phi(\boldsymbol{x}) = \boldsymbol{x}, \quad k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$$

- Kernel trick: **if we have an algorithm formulated such that x enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel.**

- Aizerman, M. A., E. M. Braverman, and L. I. Rozonoer (1964). The probability problem of pattern recognition learning and the method of potential functions. *Automation and Remote Control* **25**, 1175–1190.
- Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *Proceedings Fifth Annual Workshop on Computational Learning Theory (COLT)*, pp. 144–152. ACM.Scholkopf *et al.*,1998).

Kernel Function

- For models based on feature space mapping $\phi(\boldsymbol{x})$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$$

- Its a symmetric function:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$$

- Linear kernel:

$$\phi(\boldsymbol{x}) = \boldsymbol{x}, \quad k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$$

- Stationary kernel: invariant to translations in x

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} - \boldsymbol{x}')$$

- Homogeneous kernel (radial basis functions):

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\|\boldsymbol{x} - \boldsymbol{x}'\|)$$

Kernel Substitution Applications

- Algorithms expressed in terms of scalar products can be reformulated using kernel substitution (kernel trick). Applications include:
  - Non-linear PCA,
  - Nearest-neighbor classifiers,
  - Kernel Fisher discriminant, etc.

- Schölkopf, B., A. Smola, and K.-R. Müller (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation* **10**(5), 1299–1319.
- Mika, S., G. Rätsch, J. Weston, and B. Schölkopf (1999). Fisher discriminant analysis with kernels. In Y. H. Hu, J. Larsen, E. Wilson, and S. Douglas (Eds.), *Neural Networks for Signal Processing IX*, pp. 41–48. IEEE.
- Roth, V. and V. Steinhage (2000). Nonlinear discriminant analysis using kernel functions. In S. A. Solla, T. K. Leen, and K. R. Müller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12. MIT Press.
- Baudat, G. and F. Anouar (2000). Generalized discriminant analysis using a kernel approach. *Neural Computation* **12**(10), 2385–2404.

Textbooks on Kernel Methods include:

- Schölkopf, B. and A. J. Smola (2002). *Learning with Kernels*. MIT Press.
- Herbrich, R. (2002). *Learning Kernel Classifiers*. MIT Press.
- Shawe-Taylor, J. and N. Cristianini (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press (slides)

Dual Representation

- Many linear models for regression and classification can be reformulated in terms of a dual representation in which the kernel function arises naturally.

- Consider a linear regression model with a regularized cost functional $(\lambda > 0)$.

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

- Set the gradient to zero:

$$\boldsymbol{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right)\boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^T\boldsymbol{a}$$

$$a_n \equiv -\frac{1}{\lambda}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right), \quad \boldsymbol{a} = \left\{a_1, \ldots, a_N\right\}^T$$

- Note that we defined $\boldsymbol{a}$ using only the 1st term in $J(\boldsymbol{w})$.

Dual Representation

$$J(\boldsymbol{w})=\frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{\phi}(x_n)-t_n\right)^2+\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w},\ \boldsymbol{w}=-\frac{1}{\lambda}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{\phi}(x_n)-t_n\right)\boldsymbol{\phi}(x_n)=\boldsymbol{\Phi}^T\boldsymbol{a}$$

- Substitute $\boldsymbol{w}$ in the cost functional:

$$J(\boldsymbol{a})=\frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\phi}(x_n)-t_n\right)^2+\frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}$$

$$=\frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T\boldsymbol{\Phi}^T\boldsymbol{a}-\boldsymbol{a}^T\boldsymbol{\Phi}\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)t_n+\frac{1}{2}\boldsymbol{t}^T\boldsymbol{t}+\frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}$$

$$=\frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}-\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{t}+\frac{1}{2}\boldsymbol{t}^T\boldsymbol{t}+\frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}$$

- Here we used from an earlier lecture

$$\boldsymbol{\Phi}=\begin{pmatrix}\boldsymbol{\phi}^T(\boldsymbol{x}_1)\\ \boldsymbol{\phi}^T(\boldsymbol{x}_2)\\ \vdots\\ \boldsymbol{\phi}^T(\boldsymbol{x}_N)\end{pmatrix}=\begin{pmatrix}\phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & .. & \phi_{M-1}(\boldsymbol{x}_1)\\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & .. & \phi_{M-1}(\boldsymbol{x}_2)\\ \vdots & \vdots & \vdots & \vdots\\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & .. & \phi_{M-1}(\boldsymbol{x}_N)\end{pmatrix},$$

$$\boldsymbol{\Phi}^T\boldsymbol{\Phi}=\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T,$$

$$\boldsymbol{\Phi}^T\boldsymbol{t}=\sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n)$$

Dual Representation: Gramm Matrix

$$J(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}$$

- We can now define the Gramm matrix (symmetric) $\boldsymbol{K}_{N\times N}$ using the kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ as follows:

$$K_{nm} = K_{mn} = \boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) \quad or \quad \boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$$

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T = \begin{pmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\boldsymbol{x}_N) \end{pmatrix} \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_2) & .. & \boldsymbol{\phi}(\boldsymbol{x}_N) \end{pmatrix}$$

- The cost function takes the dual representation form:

$$J(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

Dual Representation

$$J(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

- Setting the gradient wrt $\boldsymbol{a}$ equal to zero:

$$\boldsymbol{a} = \left(\boldsymbol{K} + \lambda \boldsymbol{I}_{N\times N}\right)^{-1}\boldsymbol{t}, \quad K_{nm} = K_{mn} = k(\boldsymbol{x}_n, \boldsymbol{x}_m) \quad or \quad \boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T \ (N \times N \ matrix)$$

- Using $\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{a}$, the prediction for a new input $\boldsymbol{x}$ is

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\Phi}^T\boldsymbol{a} = \boldsymbol{k}(\boldsymbol{x})^T\left(\boldsymbol{K} + \lambda\boldsymbol{I}_{N\times N}\right)^{-1}\boldsymbol{t}$$

where

$$\boldsymbol{k}(\boldsymbol{x})^T = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\Phi}^T = \left(\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}_1) \quad \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}_2) \quad .. \quad \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}_N)\right)$$

$$= \left(k(\boldsymbol{x}_1, \boldsymbol{x}) \quad k(\boldsymbol{x}_2, \boldsymbol{x}) \quad .. \quad k(\boldsymbol{x}_N, \boldsymbol{x})\right)$$

- We now invert an $N \times N$ matrix instead of $M \times M$ (to compute $\boldsymbol{w}$). Everything is in terms of kernels (many choices here) but often $N >> M$.

Summary of the Dual Representation

- The dual formulation allows the solution to the problem to be expressed entirely in terms of the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$.

- The prediction at $\boldsymbol{x}$ is given by a linear combination of the target values from the training set.

$$y(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T \left(\boldsymbol{K} + \lambda \boldsymbol{I}_{N \times N} \right)^{-1} \boldsymbol{t}$$

$$\boldsymbol{k}(\boldsymbol{x})^T = \left(k(\boldsymbol{x}_1, \boldsymbol{x}) \quad k(\boldsymbol{x}_2, \boldsymbol{x}) \quad .. \quad k(\boldsymbol{x}_N, \boldsymbol{x}) \right)$$

- We already have seen this result in a different form in the linear regression set of notes.

- Because we now have to invert a larger matrix $N \times N$, the formulation does not seem practical.

- However, we now have the choice of working with kernel functions rather than the basis functions $\boldsymbol{\phi}(\boldsymbol{x})$. The future space can even be infinite dimensional.

Dual of the Dual Representation

- It is not difficult to see that the dual of the dual formulation is given by our original least squares representation for $\boldsymbol{w}$.

- Note that $J(\boldsymbol{a})$ can be written in terms of $\boldsymbol{Ka}, \boldsymbol{K} = \boldsymbol{\Phi\Phi}^T$

$$J(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi\Phi}^T\boldsymbol{\Phi\Phi}^T\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{\Phi\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi\Phi}^T\boldsymbol{a} =$$

$$= \frac{1}{2}\left(\boldsymbol{Ka} - \boldsymbol{t}\right)^T\left(\boldsymbol{Ka} - \boldsymbol{t}\right) + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{Ka}$$

- Recall that:

$$\boldsymbol{K} = \boldsymbol{\Phi\Phi}^T = \begin{pmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\boldsymbol{x}_N) \end{pmatrix} \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_2) & .. & \boldsymbol{\phi}(\boldsymbol{x}_N) \end{pmatrix}$$

- Since $N >> M$, $\boldsymbol{K}$ is rank deficient – $N - M$ eigenvectors of $\boldsymbol{K}$ have zero eigenvalues. We can decompose $\boldsymbol{a}$ as: $\boldsymbol{a} = \boldsymbol{a}_{\|} + \boldsymbol{a}_{\perp}$, $\boldsymbol{a}_{\perp}^T\boldsymbol{a}_{\|} = 0$ and $\boldsymbol{K}\boldsymbol{a}_{\perp} = \boldsymbol{0}$. Thus $\boldsymbol{a}_{\perp}$ is not determined by $J(\boldsymbol{a})$. One can set $\boldsymbol{a}_{\perp} = 0$, and thus

$$\boldsymbol{a} = \boldsymbol{a}_{\|} \in span\boldsymbol{K}\ (\boldsymbol{K} = \boldsymbol{\Phi\Phi}^T) \Rightarrow \boldsymbol{a} = \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^T\boldsymbol{v}\right) \Rightarrow \boldsymbol{a} = \boldsymbol{\Phi}\mathrm{u}$$

$\boldsymbol{u}$

Dual of the Dual Representation

- The cost functional becomes:

$$J(\boldsymbol{u}) = \frac{1}{2}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{u} - \boldsymbol{t}\right)^T\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{u} - \boldsymbol{t}\right) + \frac{\lambda}{2}\boldsymbol{u}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{u}$$

- Since $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ has full rank, we can introduce an equivalent parametrization as

$$\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{u}$$

- The cost functional now becomes:

$$J(\boldsymbol{u}) = \frac{1}{2}\left(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}\right)^T\left(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}\right) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w}$$

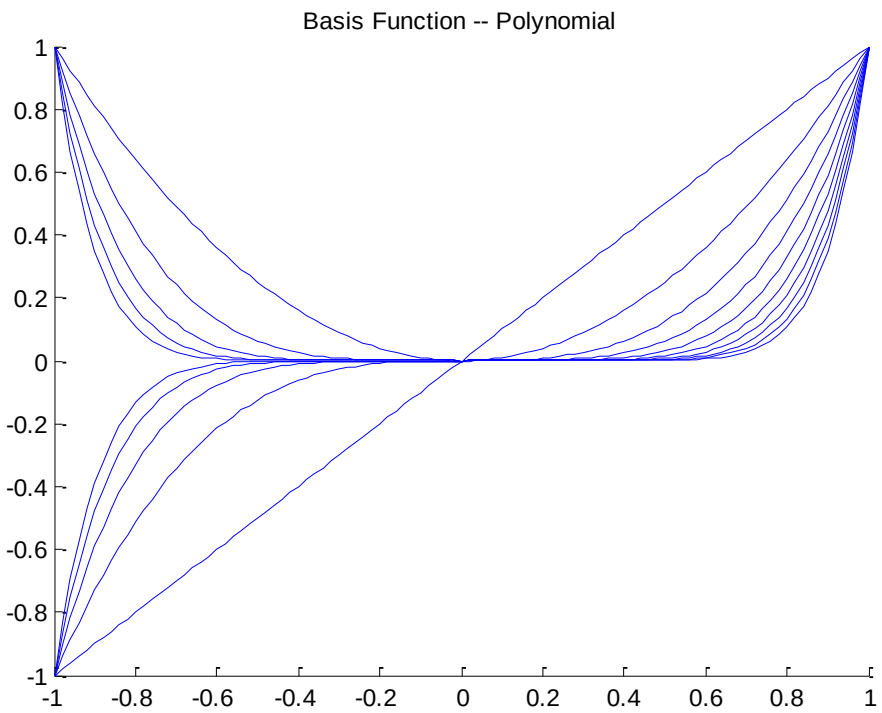
- This is our original least squares problem. So the proof is complete.

Constructing Kernels From Basis Functions

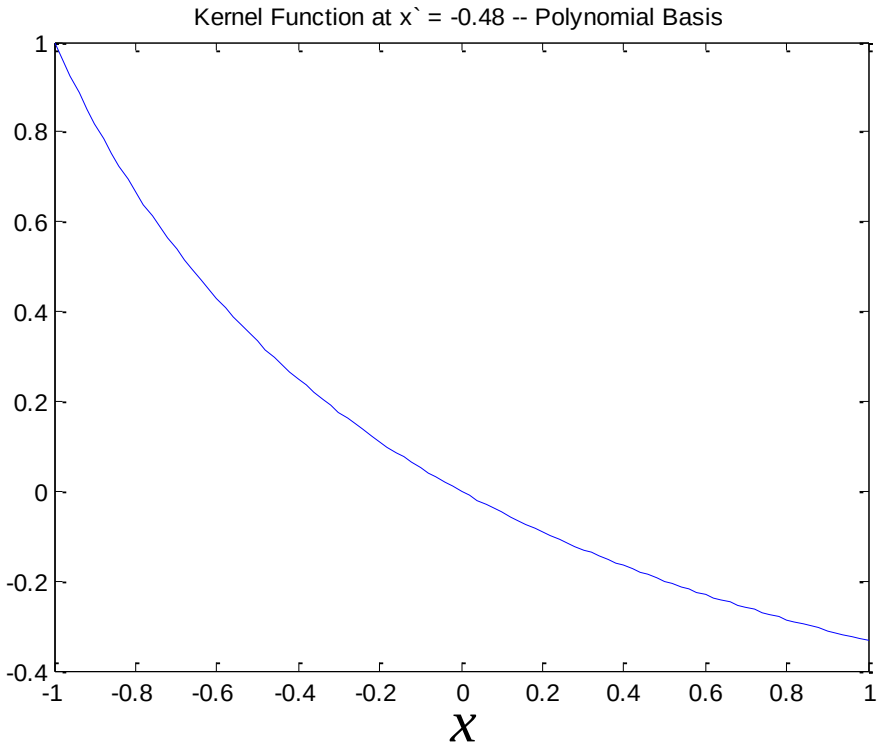
- Choose feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$, then the kernel (in 1D) is given as:

$$k(x, x') = \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$

Polynomials



Basis



Kernel $k(x, x')$

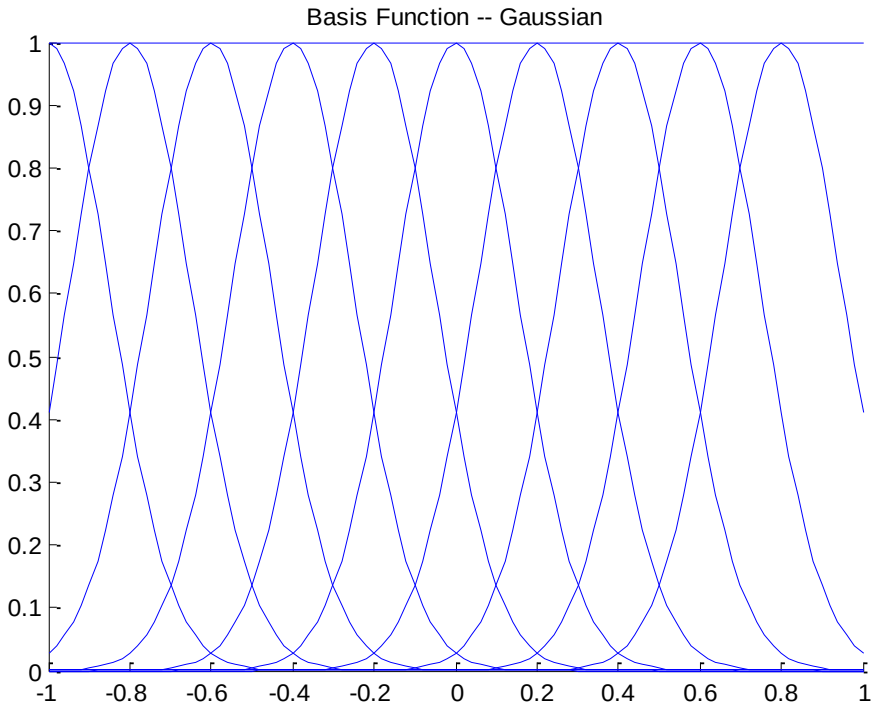
MatLab Code

Constructing Kernels From Basis Functions

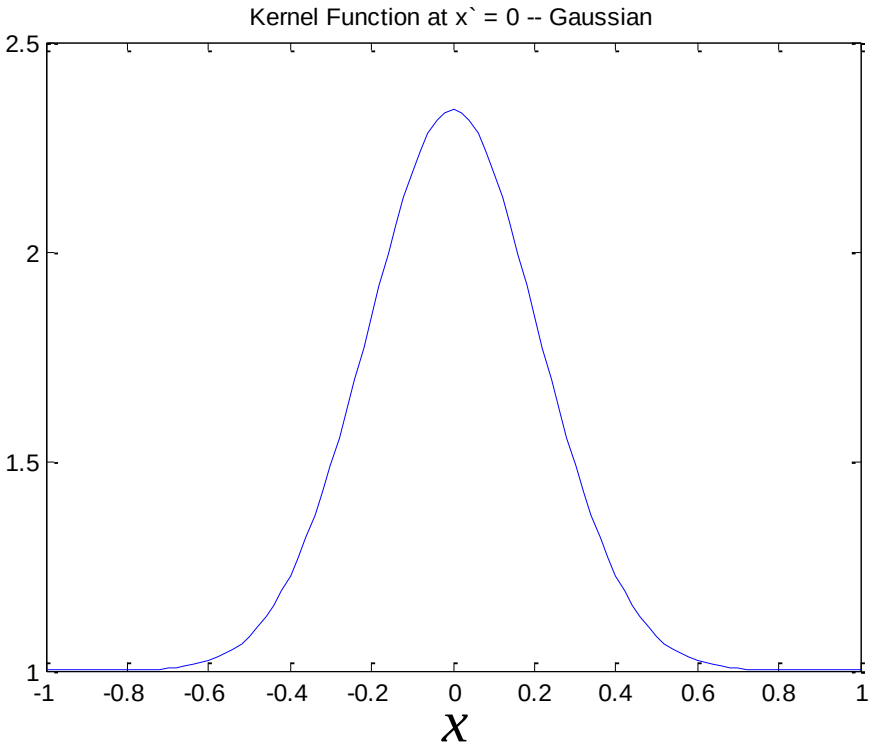
- Choose feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$, then the kernel (in 1D) is given as:

$$k(x, x') = \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$

Gaussians



Basis



Kernel $k(x, x')$

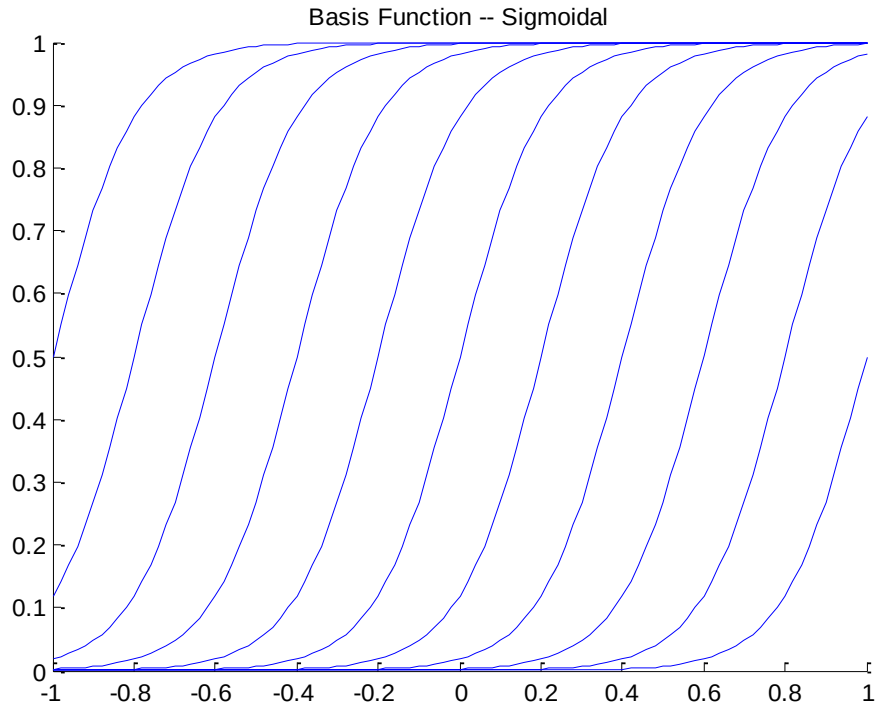
MatLab Code

Constructing Kernels From Basis Functions

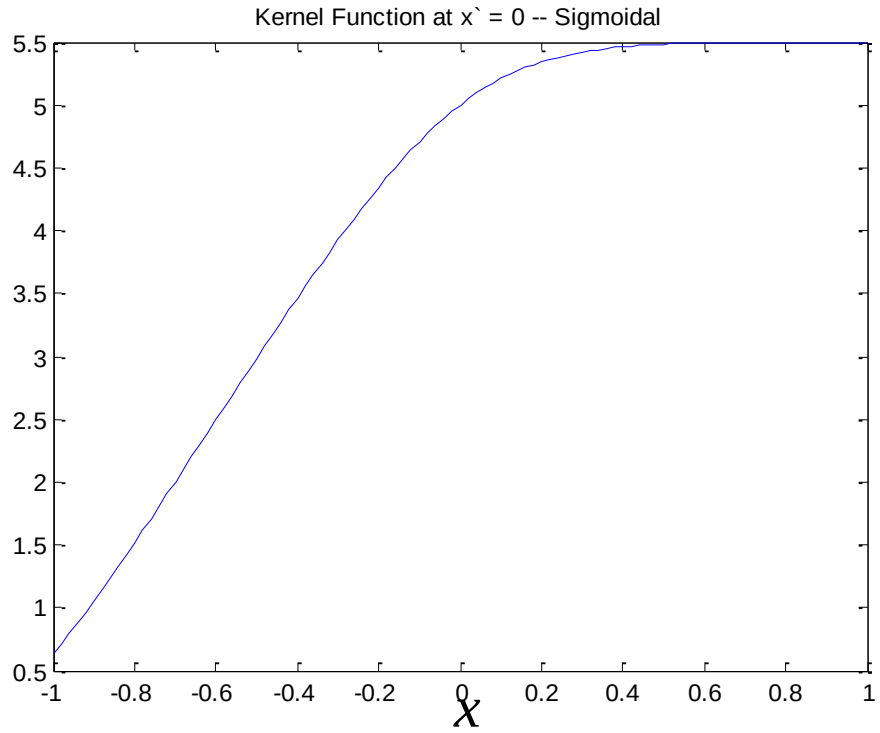
- Choose feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$, then the kernel (in 1D) is given as:

$$k(x, x') = \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$

Logistic Sigmoid



Basis



Kernel $k(x, x')$

MatLab Code

Constructing Kernels Directly

- When constructing the kernel functional directly verify its validity
  - **it should correspond to a scalar product in some feature space**
- Simple example

$$k(\boldsymbol{x},\boldsymbol{z}) = \left(\boldsymbol{x}^T\boldsymbol{z}\right)^2$$

- In the 2-D case this corresponds to

$$k(\boldsymbol{x},\boldsymbol{z}) = \left(x_1 z_1 + x_2 z_2\right)^2 = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{z})$$

where

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)^T$$

- To **test validity without having to construct $\boldsymbol{\phi}(\boldsymbol{x})$ explicitly, one can check if $K_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ is semidefinite:**

Function $k(\boldsymbol{x_n}, \boldsymbol{x}_m)$ is a valid kernel <–> $K \geq 0$ (Gram matrix) for all $\{x_n\}$

- Shawe-Taylor, J. and N. Cristianini (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press (slides)

Kernel Engineering

- Build kernels using simple kernels as building blocks. Given valid kernels $k_1(\boldsymbol{x}, \boldsymbol{x}'), k_2(\boldsymbol{x}, \boldsymbol{x}')$, the following kernels are valid:

$$k(\boldsymbol{x}, \boldsymbol{x}') = ck_1(\boldsymbol{x}, \boldsymbol{x}'), \qquad k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})k_1(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x}')$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = q\left(k_1(\boldsymbol{x}, \boldsymbol{x}')\right), \qquad k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(k_1(\boldsymbol{x}, \boldsymbol{x}')\right)$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}'), \qquad k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}')$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_3(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')), \qquad k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}'$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_a(\boldsymbol{x}_a, \boldsymbol{x}_a') + k_b(\boldsymbol{x}_b, \boldsymbol{x}_b'), \qquad k(\boldsymbol{x}, \boldsymbol{x}') = k_a(\boldsymbol{x}_a, \boldsymbol{x}_a')k_b(\boldsymbol{x}_b, \boldsymbol{x}_b')$$

with corresponding conditions:

$c > 0$, $f(.)$ arbitrary, $q(.)$ polynomial with ≥ 0 coefficients,

$\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^M$, $k_3(.,.)$ valid kernel in $\mathbb{R}^M$, A sym. ≥ 0,

$\boldsymbol{x} = (\boldsymbol{x}_a, \boldsymbol{x}_b)$, k_a, k_b valid kernels

Combining Kernels: Kernel Engineering

- If $k_1(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel, so is $k(\boldsymbol{x}, \boldsymbol{x}') = ck_1(\boldsymbol{x}, \boldsymbol{x}'), c > 0$

$$k(\boldsymbol{x}, \boldsymbol{x}') = ck_1(\boldsymbol{x}, \boldsymbol{x}') = c\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') = \boldsymbol{u}(\boldsymbol{x})^T \boldsymbol{u}(\boldsymbol{x}'), \text{ where } \boldsymbol{u}(\boldsymbol{x}) = \sqrt{c}\boldsymbol{\phi}(\boldsymbol{x})$$

- If $k_1(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel so is $k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})k_1(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x}')$ for $f(.)$ arbitrary.

$$k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})k_1(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x}') = f(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')f(\boldsymbol{x}') = \boldsymbol{u}(\boldsymbol{x})^T \boldsymbol{u}(\boldsymbol{x}'), \text{ where}$$

$$\boldsymbol{u}(\boldsymbol{x}) = f(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})$$

- If $k_1(\boldsymbol{x}, \boldsymbol{x}'), k_2(\boldsymbol{x}, \boldsymbol{x}')$ are valid kernels so is $k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$
  Indeed, note that the Gram matrix $\boldsymbol{K}$ with elements $k(x_n, x_m)$ for this kernel is ≥ 0.

$$\forall \boldsymbol{a},\ \boldsymbol{a}^T \boldsymbol{K}\boldsymbol{a} = \boldsymbol{a}^T \boldsymbol{K}_1\boldsymbol{a} + \boldsymbol{a}^T \boldsymbol{K}_2\boldsymbol{a} \geq 0, \text{ since } \boldsymbol{K}_1, \boldsymbol{K}_2 \geq 0$$

Kernel Examples

- Polynomial kernels:
  - $k(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}'\right)^2$: $\boldsymbol{\phi}(\boldsymbol{x})$ contains only terms of degree 2
  - $k(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}' + \boldsymbol{c}\right)^2, c > 0$: $\boldsymbol{\phi}(\boldsymbol{x})$ contains constant, linear and order 2 terms
  - $k(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}'\right)^M$: contains all monomials of order M

    Example: For $\boldsymbol{x}$ and $\boldsymbol{x}'$ images, $k(x, x')$ is a weighted sum of all possible products of M pixels of image $\boldsymbol{x}$ with M pixels of image $\boldsymbol{x}'$.
  - $k(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}' + \boldsymbol{c}\right)^M, c > 0$: contains all terms up to order M

Gaussian Kernel

- Gaussian kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2}{2\sigma^2} \right)$$

- To see that this is a valid kernel note:

$$\left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2 = \boldsymbol{x}^T \boldsymbol{x} + \boldsymbol{x}'^T \boldsymbol{x}' - 2\boldsymbol{x}^T \boldsymbol{x}'$$

and write it as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\sigma^2} \left(\boldsymbol{x}^T \boldsymbol{x} + \boldsymbol{x}'^T \boldsymbol{x}' - 2\boldsymbol{x}^T \boldsymbol{x}' \right) \right)$$

$$= \underbrace{\exp\left(-\frac{\boldsymbol{x}^T \boldsymbol{x}}{2\sigma^2} \right)}_{f(\boldsymbol{x})} \underbrace{\exp\left(\frac{\boldsymbol{x}^T \boldsymbol{x}'}{\sigma^2} \right)}_{k_1(\boldsymbol{x}, \boldsymbol{x}')} \underbrace{\exp\left(-\frac{\boldsymbol{x}'^T \boldsymbol{x}'}{2\sigma^2} \right)}_{f(\boldsymbol{x}')}$$

- "Taylor expanding" the middle exponential, we can see that the Gaussian kernel can be expressed as the inner product of an infinite dimensional feature vector.

Gaussian Kernel: non-Euclidean Distance

- Gaussian kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\sigma^2} \left(\boldsymbol{x}^T \boldsymbol{x} + \boldsymbol{x}'^T \boldsymbol{x}' - 2\boldsymbol{x}^T \boldsymbol{x}' \right) \right)$$

- With the kernel trick, we can replace the Euclidean distance with any non-linear kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\sigma^2} \left(\kappa(\boldsymbol{x}, \boldsymbol{x}) + \kappa(\boldsymbol{x}', \boldsymbol{x}') - 2\kappa(\boldsymbol{x}, \boldsymbol{x}') \right) \right)$$

Example: Symbolic Inputs

- The inputs can be symbolic, graphs, sets, strings, text, etc.

- For example, fix a set D and consider all of its subsets ($2^{|D|}$). Is A_1 and A_2 are two subsets of D, we can define the kernel as:

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

  where $|A_1 \cap A_2|$ is the number of elements in the intersection $A_1 \cap A_2$

- $\phi(D)$ is defined to map to a vector of $2^{|D|}$ 1's, one for each possible subset of D, including D itself as well as the empty set.

- For $A \subset D$, $\phi(A)$ will have 1's in all positions that correspond to subsets of A and 0's in all other positions. Therefore, $\phi(A_1)^T \phi(A_2)$ will count the number of subsets shared by A_1 and A_2.

- This can also be obtained by counting the number of elements in the intersection of A_1 and A_2, and then raising 2 to this number, which is exactly $k(A_1, A_2) = 2^{|A_1 \cap A_2|}$

Probabilistic Generative Models

- Use a generative model to define a kernel and then use this kernel in a discriminative approach.
- Kernel for generative model $p(x)$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = p(\boldsymbol{x})\, p(\boldsymbol{x}')$$

- This is a valid kernel function since it is an inner product in the 1D feature space defined by the mapping $p(\boldsymbol{x})$.
- It says that two inputs $\boldsymbol{x}$ and $\boldsymbol{x}$' are similar if they both have high probabilities.

- Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, Computer Science Department.
- Lasserre, J., C. M. Bishop, and T. Minka (2006). Principled hybrids of generative and discriminative models. In *Proceedings 2006 IEEE Conference on Computer Vision and Pattern Recognition, New York*.

Probabilistic Generative Models

- Using $k(\boldsymbol{x}, \boldsymbol{x}') = p(\boldsymbol{x})p(\boldsymbol{x}')$ and the kernel engineering formulas

$$k(\boldsymbol{x}, \boldsymbol{x}') = ck_1(\boldsymbol{x}, \boldsymbol{x}'), c > 0$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$$

we can generalize as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_i p(\boldsymbol{x} \mid i)\, p(\boldsymbol{x}' \mid i) p(i)$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = \int p(\boldsymbol{x} \mid \boldsymbol{z}) p(\boldsymbol{x}' \mid \boldsymbol{z})\, p(\boldsymbol{z}) d\boldsymbol{z}$$

i and $\boldsymbol{z}$ are latent variables.

- x and x' will give large value of $k(x, x')$ if they have significant probability under a range of components.

Kernel Function in HMMs

- Now consider an observation is given by a sequence $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L\}$.
- A popular generative model for sequences is the Hidden Markov model (HMM), which expresses the distribution $p(\boldsymbol{X})$ as a marginalization over a corresponding sequence of hidden states $\boldsymbol{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L\}$.
- We define a kernel function measuring the similarity of two sequences $\boldsymbol{X}$ and $\boldsymbol{X}'$ by extending the earlier mixture representation

$$k(\boldsymbol{X}, \boldsymbol{X}') = \sum_{\boldsymbol{Z}} p(\boldsymbol{X} \mid \boldsymbol{Z}) p(\boldsymbol{X}' \mid \boldsymbol{Z}) p(\boldsymbol{Z})$$

- Note that both observed sequences $\boldsymbol{X}, \boldsymbol{X}'$ are generated by the same hidden sequence $\boldsymbol{Z}$.

Fisher Kernel

- Consider a parametric generative model $p(\boldsymbol{x}|\boldsymbol{\theta})$. We want a kernel that measures the similarity of two input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ induced by this distribution.
- The Fisher score is introduced defining a vector in future space

$$g\left(\boldsymbol{\theta}, x\right) = \nabla_{\theta} \ln p\left(\boldsymbol{x} \mid \boldsymbol{\theta}\right)$$

- The Fisher kernel is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}') = g(\boldsymbol{\theta}, \boldsymbol{x})^T \mathbf{F}^{-1} g(\boldsymbol{\theta}, \boldsymbol{x}')$$

The Fisher information matrix $\boldsymbol{F}$ (considers the geometry of the parameter/future space) is given as the expectation under $p\left(\boldsymbol{x} \mid \boldsymbol{\theta}\right)$

$$\mathbf{F} = \mathbb{E}_{\boldsymbol{x}}\left[g(\boldsymbol{\theta}, \boldsymbol{x}) g(\boldsymbol{\theta}, \boldsymbol{x})^T \right]$$

- Using $\boldsymbol{F}$ makes the kernel invariant under $\boldsymbol{\theta} \rightarrow \psi(\boldsymbol{\theta})$ where $\psi(\boldsymbol{\theta})$ is non-linear reparametrization that is invertible & differentiable.

- Jaakkola, T. S. and D. Haussler (1999). Exploiting generative models in discriminative classifiers. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in Neural Information Processing Systems*, Volume 11. MIT Press.
- Amari, S. I. (1998). Natural gradient works efficiently in learning. *Neural Computation* **10**, 251–276.

Fisher Kernel

$$\mathbf{F} = \mathbb{E}_{\boldsymbol{x}}\left[g(\boldsymbol{\theta}, \boldsymbol{x}) g(\boldsymbol{\theta}, \boldsymbol{x})^T \right]$$

- We often approximate the information matrix with a sample average:

$$\boldsymbol{F} \approx \frac{1}{N} \sum_{n=1}^{N} g(\theta, \boldsymbol{x}_n) g(\theta, \boldsymbol{x}_n)^T$$

- This is the covariance matrix of the Fisher scores, and so the Fisher kernel corresponds to a whitening of these scores.

$$k(\boldsymbol{x}, \boldsymbol{x}') = g(\boldsymbol{\theta}, \boldsymbol{x})^T \mathbf{F}^{-1} g(\boldsymbol{\theta}, \boldsymbol{x}')$$

- More simply, we can just omit the Fisher information matrix altogether and use the non-invariant kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = g(\boldsymbol{\theta}, \boldsymbol{x})^T g(\boldsymbol{\theta}, \boldsymbol{x}')$$

- Hofmann, T. (2000). Learning the similarity of documents: an information-geometric approach to document retrieval and classification. In S. A. Solla, T. K. Leen, and K. R. M¨uller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12, pp. 914–920. MIT Press.

Fisher Kernel: Example

- Let us compute the Fisher kernel for the Gaussian distribution with fixed covariance $\boldsymbol{S}$:

$$p(\boldsymbol{x} \mid \boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{S})$$

- The Fisher score is given:

$$g(\boldsymbol{\mu}, \boldsymbol{x}) = \nabla_{\mu} \ln p(\boldsymbol{x} \mid \boldsymbol{\mu}) = \boldsymbol{S}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$$

- The Fisher information matrix $\boldsymbol{F}$ is then:

$$\mathbf{F} = \mathbb{E}_{x}\left[g(\boldsymbol{\mu}, \boldsymbol{x}) g(\boldsymbol{\mu}, \boldsymbol{x})^{T} \right] = \boldsymbol{S}^{-1} \underbrace{\mathbb{E}_{x}\left[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^{T} \right]}_{\boldsymbol{S}} \boldsymbol{S}^{-1} = \boldsymbol{S}^{-1}$$

- The Fisher kernel is then given as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = g(\boldsymbol{\mu}, \boldsymbol{x})^{T} \mathbf{F}^{-1} g(\boldsymbol{\mu}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{\mu})^{T} \boldsymbol{S}^{-1} (\boldsymbol{x}' - \boldsymbol{\mu})$$

- This is the Mahalanobis distance!

Sigmoidal Kernel

- Another kernel function is the sigmoidal kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(a\boldsymbol{x}^T \boldsymbol{x}' + b)$$

- Unfortunately the Gram matrix in general is not positive semidefinite.
- This kernel was used because it gives kernel expansions such as the SVM a superficial resemblance to neural network models.
- As we shall see later in this lecture, in the limit of an infinite number of basis functions, a Bayesian neural network with an appropriate prior reduces to a Gaussian process, thereby providing a link between neural networks and kernel methods.

- Vapnik, V. N. (1995). *The nature of statistical learning theory*. Springer

Radial Basis Functions

- By definition

$$\phi_j(\boldsymbol{x}) = h\left(\left\|\boldsymbol{x} - \boldsymbol{\mu}_j\right\|\right)$$

where $\left\|\boldsymbol{x} - \boldsymbol{\mu}_j\right\|$ is a radial distance from a center $\boldsymbol{\mu}_j$.

- Originally were introduced for the problem of exact interpolation $f(\boldsymbol{x}_n) = t_n, n = 1, \ldots, N$

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} w_n h\left(\left\|\boldsymbol{x} - \boldsymbol{x}_n\right\|\right)$$

- There are as many unknowns w_n as target values t_n. Thus least squares leads to exact interpolation.

- If the target values t_n are noisy, exact interpolation overfits.

- Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation: a review. In J. C. Mason and M. G. Cox (Eds.), *Algorithms for Approximation*, pp. 143–167. Oxford University Press.

Radial Basis: Regularized Least Squares

- For a sum-of-squares error function with a regularizer defined in terms of a differential operator, the optimal solution is given by an expansion in the Green's functions of the operator (analogous to eigenvectors of a discrete matrix) again with one basis function centered on each data point.

- Green's functions for an isotropic differential operator as regularizer lead to radial basis functions (depending only on the distance from the data points)

- Interpolation is not exact due to the presence of the regularizer.

- Poggio, T. and F. Girosi (1990). Networks for approximation and learning. *Proceedings of the IEEE* **78**(9), 1481–1497.
- Bishop, C. M. (1995a). *Neural Networks for Pattern Recognition*. Oxford University Press

Interpolation with Noisy Inputs

- Consider the interpolation problem with noisy inputs rather than target values:

$$E = \frac{1}{2}\sum_{n=1}^{N}\int \left\{ y\left(\boldsymbol{x}_n + \boldsymbol{\xi}\right) - t_n \right\}^2 \nu(\boldsymbol{\xi}) d\boldsymbol{\xi}$$

- We can optimize with respect to the function $y\left(\boldsymbol{x}\right)$ to obtain:

$$y\left(\boldsymbol{x}\right) = \sum_{n=1}^{N} t_n h\left(\boldsymbol{x} - \boldsymbol{x}_n\right)$$

- The basis functions are given as (Nadaraya-Watson model):

$$h\left(\boldsymbol{x} - \boldsymbol{x}_n\right) = \frac{\nu\left(\boldsymbol{x} - \boldsymbol{x}_n\right)}{\sum_{n=1}^{N} \nu\left(\boldsymbol{x} - \boldsymbol{x}_n\right)}$$

- If noise distribution $\nu(\boldsymbol{\xi})$ is isotropic (function of $\|\boldsymbol{\xi}\|$), these basis functions would be radial.

- Bishop, C. M. (1995a). *Neural Networks for Pattern Recognition*. Oxford University Press
- Webb, A. R. (1994). Functional approximation by feed-forward networks: a least-squares approach to generalisation. *IEEE Transactions on Neural Networks* **5**(3), 363–371.

Radial Basis Functions: Proof

- We take a variation of $y(\boldsymbol{x})$: $y(\boldsymbol{x}) \rightarrow y(\boldsymbol{x}) + e\eta(\boldsymbol{x})$. Then:

$$E[y+\varepsilon\eta] = \frac{1}{2}\sum_{n=1}^{N}\int\left\{y(\boldsymbol{x}_n+\boldsymbol{\xi})+\varepsilon\eta(\boldsymbol{x}_n+\boldsymbol{\xi})-t_n\right\}^2\nu(\boldsymbol{\xi})d\boldsymbol{\xi}$$

- We expand in ε and set the coefficient in ε (functional 1st derivative) to zero:

$$\sum_{n=1}^{N}\int\left\{y(\boldsymbol{x}_n+\boldsymbol{\xi})-t_n\right\}\eta(\boldsymbol{x}_n+\boldsymbol{\xi})\nu(\boldsymbol{\xi})d\boldsymbol{\xi} = 0$$

- Since this must be true for any variation $\eta(\boldsymbol{x})$, we choose: $\eta(\boldsymbol{x}) = \delta(\boldsymbol{x}-\boldsymbol{z})$. This leads to:

$$\sum_{n=1}^{N}\int\left\{y(\boldsymbol{x}_n+\boldsymbol{\xi})-t_n\right\}\delta(\boldsymbol{x}_n+\boldsymbol{\xi}-\boldsymbol{z})\nu(\boldsymbol{\xi})d\boldsymbol{\xi} = \sum_{n=1}^{N}\left\{y(\boldsymbol{z})-t_n\right\}\nu(\boldsymbol{z}-\boldsymbol{x}_n) = 0 \Rightarrow$$

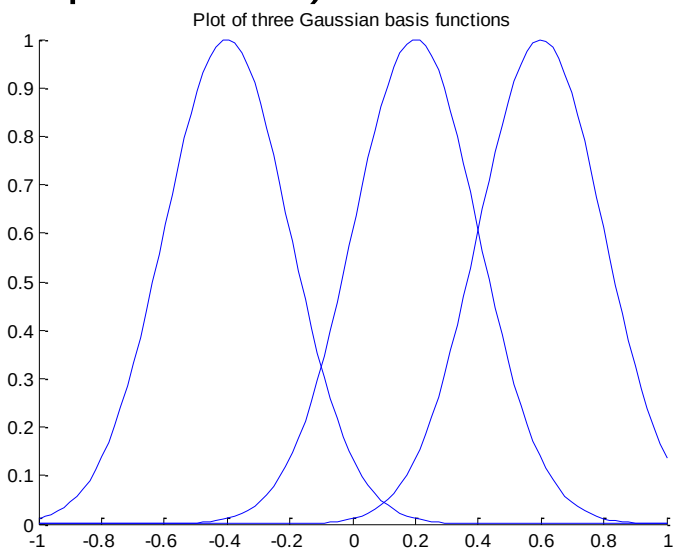
$$y(\boldsymbol{z}) = \sum_{n=1}^{N} t_n\left(\nu(\boldsymbol{z}-\boldsymbol{x}_n)\Big/\sum_{l=1}^{N}\nu(\boldsymbol{z}-\boldsymbol{x}_l)\right)$$

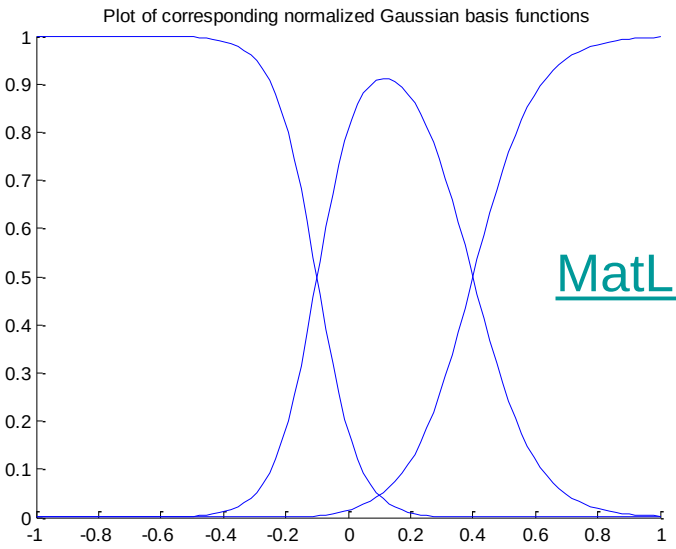
Nadaraya-Watson Model

- The functions in the Nadaraya-Watson model are normalized:

$$h(\boldsymbol{x} - \boldsymbol{x}_n) = \frac{\nu(\boldsymbol{x} - \boldsymbol{x}_n)}{\sum_{n=1}^{N} \nu(\boldsymbol{x} - \boldsymbol{x}_n)}, \qquad \sum_n h(\boldsymbol{x} - \boldsymbol{x}_n) = 1$$

- It is useful to have normalized basis functions to avoid regions in an input space where all of the basis functions take small values (leading to small predictions or predictions controlled solely by the bias parameter).





MatLab Code

Radial Basis For Regression

- In the application of kernel density estimation to regression, there is one basis function associated with every data point, and the corresponding model can be computationally costly to evaluate when making predictions for new data points.
- Models have therefore been proposed using the expansion in radial basis functions but where the number M of basis functions is smaller than the number N of data points.
- Typically, the number of basis functions, and the centers $\boldsymbol{\mu}_i$ are determined based on the input data $\{\boldsymbol{x}_n\}$ alone. The basis functions are kept fixed and the coefficients $\{w_i\}$ are determined by least squares.

- Broomhead, D. S. and D. Lowe (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems* **2**, 321–355.
- Moody, J. and C. J. Darken (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation* **1**(2), 281–294.
- Poggio, T. and F. Girosi (1990). Networks for approximation and learning. *Proceedings of the IEEE* **78**(9), 1481–1497.

Reducing the Size of the Basis

- To reduce the algorithmic cost, we can keep the number of basis functions *M* smaller than input data size N
- Centers locations μ_i are determined based on the input data $\{x_n\}$ alone
- Coefficients $\{w_i\}$ are determined by least squares
- For the choice of centers:
  - Random subset of the training data points
  - Orthogonal least squares (sequential) – the next data point to be chosen as a basis function center gives the greatest error reduction
  - Using clustering algorithms (K-means) – basis functions centers no longer coincide with the training data points.

- Chen, S., C. F. N. Cowan, and P. M. Grant (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks* **2**(2), 302–309.

Parzen Density Estimator

- We have seen the prediction of the linear regression model - linear combination of t_n with `equivalent kernel' values

- The same result is obtained from the Parzen density estimator. $f(\boldsymbol{x}, t)$ is the component density function centered at each training data point

$$p(\boldsymbol{x}, t) = \frac{1}{N} \sum_{n=1}^{N} f(\boldsymbol{x} - \boldsymbol{x}_n, t - t_n)$$

- Regression function

$$y(\boldsymbol{x}) = \mathbb{E}[t \mid \boldsymbol{x}] = \int_{-\infty}^{+\infty} t p(t \mid \boldsymbol{x}) dt =$$

$$= \frac{\int t p(\boldsymbol{x}, t) dt}{\int p(\boldsymbol{x}, t) dt} = \frac{\sum_n \int t f(\boldsymbol{x} - \boldsymbol{x}_n, t - t_n) dt}{\sum_m \int f(\boldsymbol{x} - \boldsymbol{x}_m, t - t_m) dt}$$

Kernel Regression: Nadaraya-Watson Model

- Assume that the component density functions have zero mean so that

$$\int_{-\infty}^{+\infty} t f(\boldsymbol{x},t)dt = 0$$

for all values of $\boldsymbol{x}$. Then by variable change

$$y(\boldsymbol{x}) = \frac{\sum_n \int (t - t_n + t_n) f(\boldsymbol{x}-\boldsymbol{x}_n, t-t_n)dt}{\sum_m \int f(\boldsymbol{x}-\boldsymbol{x}_m, t-t_m)dt} = \frac{\sum_n g(\boldsymbol{x}-\boldsymbol{x}_n)t_n}{\sum_m g(\boldsymbol{x}-\boldsymbol{x}_m)} =$$

$$= \sum k(\boldsymbol{x},\boldsymbol{x}_n)t_n$$

with $g(\boldsymbol{x}) = \int_{-\infty}^{+\infty} f(\boldsymbol{x},t)dt$ and $k(\boldsymbol{x},\boldsymbol{x}_n)$ given by

$$k(\boldsymbol{x},\boldsymbol{x}_n) = \frac{g(\boldsymbol{x}-\boldsymbol{x}_n)}{\sum_m g(\boldsymbol{x}-\boldsymbol{x}_m)}$$

Kernel Regression: Nadaraya-Watson Model

- For localized kernels, this regression model gives more weight to the data points $\boldsymbol{x}_n$ close to $\boldsymbol{x}$.

$$y(\boldsymbol{x}) = \sum_n k(\boldsymbol{x}, \boldsymbol{x}_n) t_n$$

$$k(\boldsymbol{x}, \boldsymbol{x}_n) = \frac{g(\boldsymbol{x} - \boldsymbol{x}_n)}{\sum_m g(\boldsymbol{x} - \boldsymbol{x}_m)}$$

- The kernel satisfies the summation constraint:

$$\sum_{n=1}^{N} k(\boldsymbol{x}, \boldsymbol{x}_n) = 1$$

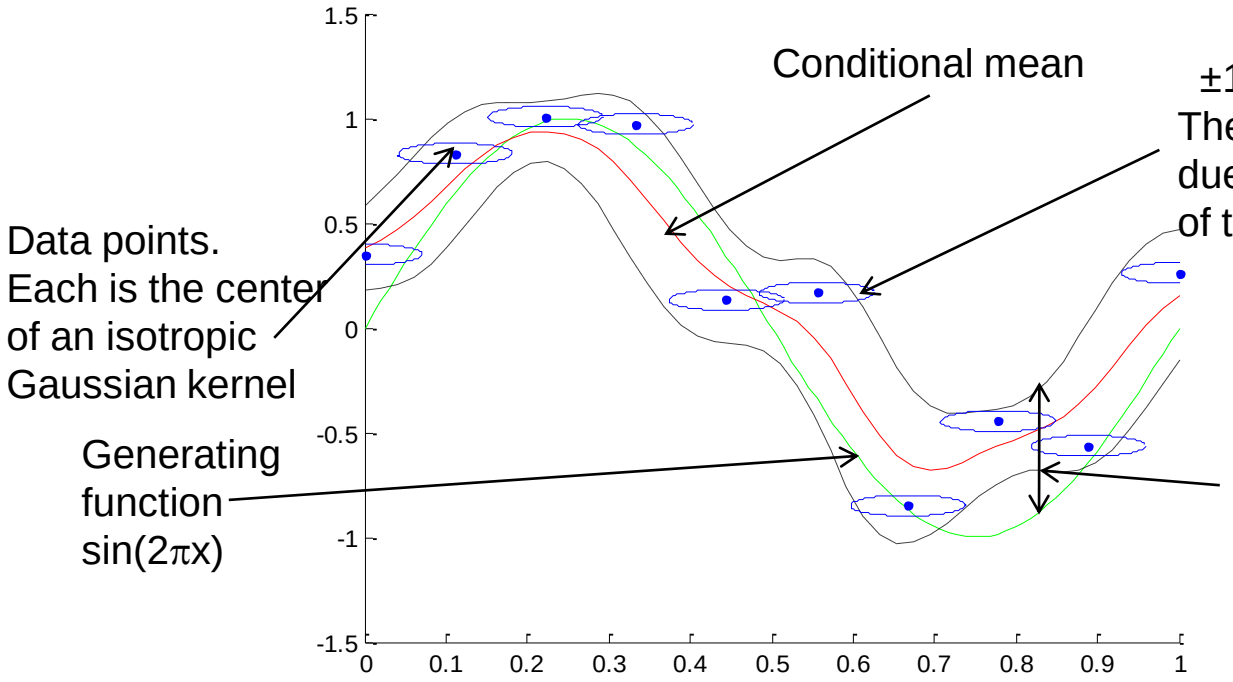
- In addition to the conditional expectation $y(\boldsymbol{x})$, the model provides the full conditional distribution:

$$p(t \mid \boldsymbol{x}) = \frac{p(t, \boldsymbol{x})}{\int p(t, \boldsymbol{x}) dt} = \frac{\sum_n f(\boldsymbol{x} - \boldsymbol{x}_n, t - t_n)}{\sum_m \int f(\boldsymbol{x} - \boldsymbol{x}_m, t - t_m) dt}$$

- For Gaussian component distributions, this is a mixture of Gaussians.

Kernel Regression: Nadaraya-Watson Model

- Single input variable x. $f(x,t)$ is a zero-mean isotropic Gaussian kernels over the variable $z=(x,t)$ with variance σ^2



MatLab Code

- Nadaraya, ´ E. A. (1964). On estimating regression. *Theory of Probability and its Applications* **9**(1), 141–142.
- Watson, G. S. (1964). Smooth regression analysis. *Sankhya: The Indian Journal of Statistics. Series A* **26**, 359–372.

Kernel Regression: Nadaraya-Watson Model

- We can use more flexible forms of Gaussian components, e.g. having different variance parameters for the input and target variables.

- We could alternatively model $p(t, \boldsymbol{x})$ using a Gaussian mixture model, trained using EM techniques and then find $p(t|\boldsymbol{x})$.

  - ✓ In this case we don't have a representation in terms of kernel functions evaluated at the training set data points.

  - ✓ However, the number of components in the mixture model can be smaller than the number of training set points, resulting in a model that is faster to evaluate for test data points.

- Ghahramani, Z. and M. I. Jordan (1994). Supervised learning from incomplete data via an EM appproach. In J. D. Cowan, G. T. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems*, Volume 6, pp. 120–127.

Gaussian Processes

- The idea is similar to linear regression with a fixed set of basis functions:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$

- However, here we forget about the parametric model $y(\boldsymbol{x}, \boldsymbol{w})$
- Instead we take an infinite number of basis functions given by a probability distribution over functions.
- This is not difficult to model since we only have to consider the values of the functions at training/test data points.
- Gaussian process models include:
  - Kriging
  - ARMA (autoregressive moving average)
  - Kalman filters
  - Radial basis function networks , etc.

Gaussian Processes

- Models equivalent to Gaussian processes have been widely studied in many different fields.
- In the geostatistics, Gaussian process regression is known as *kriging*.
- Similarly, ARMA (autoregressive moving average) models
- Kalman filters and radial basis function networks can all be viewed as forms of Gaussian process models.

- Cressie, N. (1993). *Statistics for Spatial Data*.Wiley.
- MacKay, D. J. C. (1998). Introduction to Gaussian processes. In C. M. Bishop (Ed.), *Neural Networks and Machine Learning*, pp. 133–166. Springer.
- Williams, C. K. I. (1999). Prediction with Gaussian processes: from linear regression to linear prediction and beyond. In M. I. Jordan (Ed.), *Learning in Graphical Models*, pp. 599–621. MIT Press.MacKay (2003)
- Rasmussen, C. E. (1996). *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*. Ph. D. thesis, University of Toronto.
- Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press.Gibbs, 1997;

Linear Regression Revisited

- Consider a model defined by linear combination of M fixed basis functions:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$

- A Gaussian prior distribution over the weight vector $\boldsymbol{w}$

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid 0, \alpha^{-1}\boldsymbol{I})$$

governed by the hyperparameter (precision) α induces then a probability distribution over functions $y(\boldsymbol{x})$.

Linear Regression Revisited

- Evaluating $y(\boldsymbol{x})$ for a set of training data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, we have a joint distribution of $y(\boldsymbol{x_1}), \ldots, y(\boldsymbol{x_N})$

$$\boldsymbol{y} = \boldsymbol{\Phi w}, \text{ with elements } y_n = y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)$$

where $\boldsymbol{\Phi}$ is the design matrix with elements $\Phi_{nk} = \boldsymbol{\phi}_k(\boldsymbol{x}_n)$

- $p(\boldsymbol{y})$ is Gaussian, and its mean and covariance can be shown to be

$$\mathbb{E}[\boldsymbol{y}] = \boldsymbol{\Phi}\mathbb{E}[\boldsymbol{w}] = 0$$

$$\operatorname{cov}[\boldsymbol{y}] = \mathbb{E}\left[\boldsymbol{y}\boldsymbol{y}^T\right] = \boldsymbol{\Phi}\mathbb{E}\left[\boldsymbol{w}\boldsymbol{w}^T\right]\boldsymbol{\Phi}^T \quad \Rightarrow \operatorname{cov}[\boldsymbol{y}] = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \boldsymbol{K}$$

where $\boldsymbol{K}$ is the Gram matrix with elements defined in terms of the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ as

$$K_{nm} = k(x_n, x_m) = \frac{1}{\alpha}\boldsymbol{\phi}(x_n)^T \boldsymbol{\phi}(x_m)$$

- Up to now we only took data points + prior but no target values.

Gaussian Processes

- The model we have seen is one example of a Gaussian process
- Gaussian process is defined as a probability distribution over functions $y(\boldsymbol{x})$ such that the set of values of y($\mathbf{x}$) evaluated at an arbitrary set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ jointly have a Gaussian distribution (in 2D, we have a Gaussian Random Field).
- Key point: **the joint distribution is defined completely by second-order statistics (mean, covariance)**
- Since usually the mean is taken to be zero, we only need the covariance, i.e., the kernel-function:

$$\mathbb{E}\left[y(\mathbf{x}_n) y(\mathbf{x}_m) \right] = k(\mathbf{x}_n, \mathbf{x}_m)$$

- **Instead of choosing (a limited number of) basis functions, we can directly choose a kernel function (which may result in an infinite number of basis functions)**

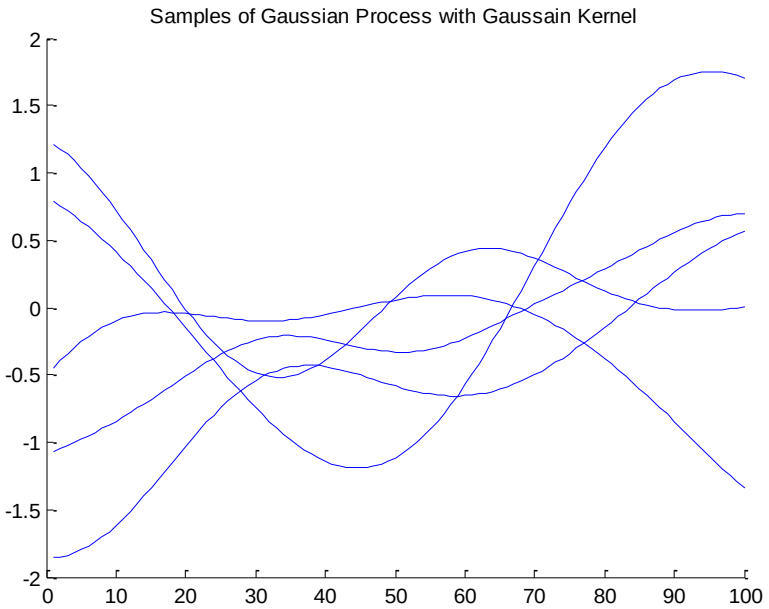
Gaussian Processes

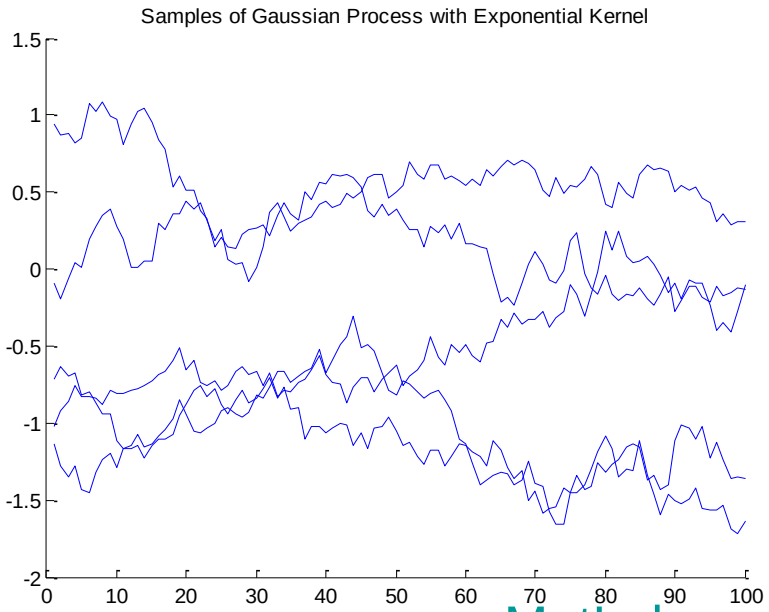
- We show examples of functions drawn from Gaussian processes for a Gaussian kernel and an exponential kernel.

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right)$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\theta\,|\,x - x'\,|\right)$$

(Ornstein-Uhlenbeck process)





MatLab code

- Uhlenbeck, G. E. and L. S. Ornstein (1930). On the theory of Brownian motion. *Phys. Rev.* **36**, 823– 841.

Gaussian Process Regression

- To use Gaussian processes for regression, we need to model noise:

$$t_n = y_n + \varepsilon_n, \quad \text{with} \quad y_n = y(\boldsymbol{x}_n)$$

- For noise processes with a Gaussian distribution we obtain

$$p(t_n \mid y_n) = \mathcal{N}\left(t_n \mid y_n, \beta^{-1}\right)$$

- The joint distribution for $\boldsymbol{t} = (t_1, t_2, ..., t_n)^T$ conditioned on $\boldsymbol{y} = (y_1, y_2, ..., y_n)^T$ is then (since the noise is assumed to be independent for each data point)

$$p(\boldsymbol{t} \mid \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{t} \mid \boldsymbol{y}, \beta^{-1}\boldsymbol{I}_N\right)$$

Gaussian Process Regression

- From the definition of a Gaussian process we have

$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K})$$

- The kernel function that determines $\boldsymbol{K}$ is chosen such that for points $\boldsymbol{x}_n, \boldsymbol{x}_m$ that are similar the corresponding values $y(\boldsymbol{x}_n), y(\boldsymbol{x}_m)$ will be stronger correlated.

- For the marginal distribution $p(\boldsymbol{t})$, we need to integrate over $\boldsymbol{y}$ (see Appendix and earlier lecture notes):

$$p(\boldsymbol{t}) = \int p(\boldsymbol{t} \mid \boldsymbol{y})\, p(\boldsymbol{y})\, d\boldsymbol{y} = \mathcal{N}(\boldsymbol{t} \mid 0, \boldsymbol{C})$$

with $$\boldsymbol{C} = \boldsymbol{K} + \beta^{-1}\boldsymbol{I} \; or \; C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}$$

Appendix: Linear Gaussian Model

- Consider a linear Gaussian model: A Gaussian marginal distribution $p(\boldsymbol{x})$ and a Gaussian conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ in which $p(\boldsymbol{y}|\boldsymbol{x})$ has a mean that is a linear function of $\boldsymbol{x}$, and a covariance which is independent of $\boldsymbol{x}$.

$$p(\mathbf{x}) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}, \Lambda^{-1}\right)$$

$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}\right)$$

- We want to find $p(\boldsymbol{y})$. We start with the joint distribution over $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ which is a Gaussian and then integrate $\boldsymbol{x}$ out.

- Finally we obtained:

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\Lambda^{-1}\mathbf{A}^{T}\right)$$

Practical Kernel Function

- One widely used kernel function for Gaussian process regression is given here:

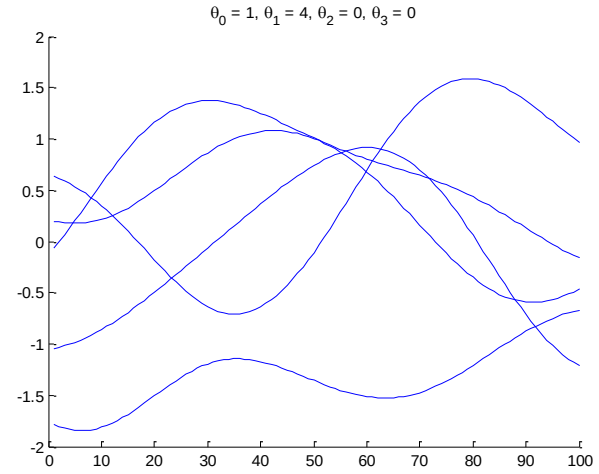
$$k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = \theta_0 \exp\left(-\frac{\theta_1}{2}\left\|\boldsymbol{x}_n - \boldsymbol{x}_m\right\|^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$$

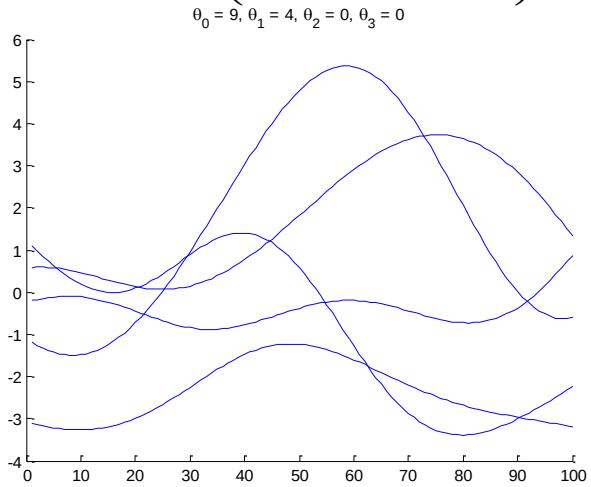
- The term involving θ_3 corresponds to a parametric model that is a linear function of the input variables.

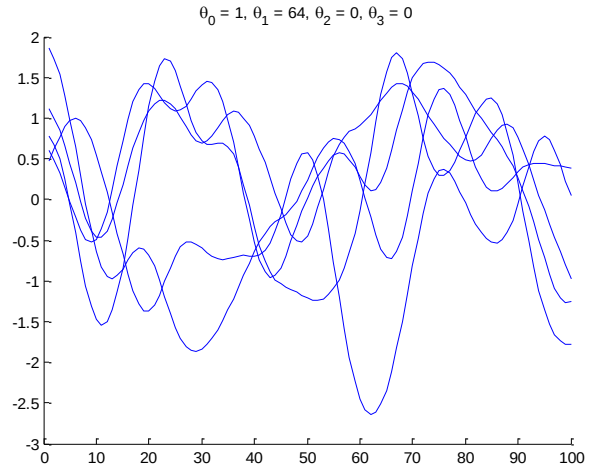
Distribution over Functions

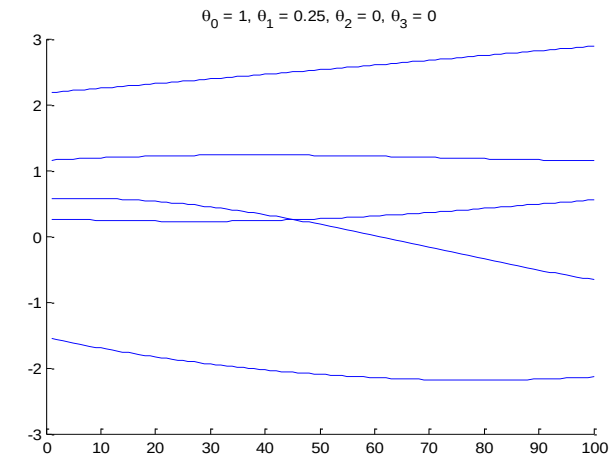
- Sample functions from prior p(y) with common kernel function

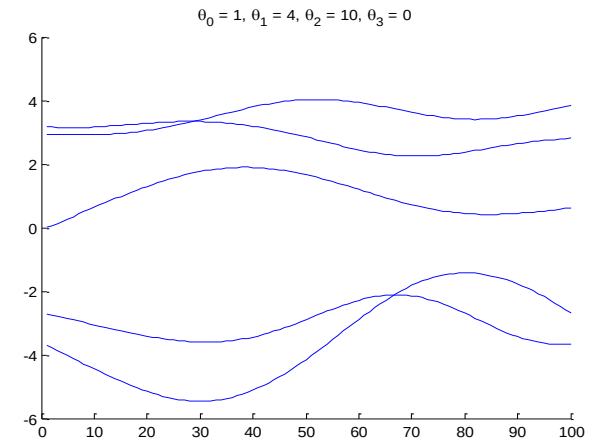
$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}\|\boldsymbol{x}_n - \boldsymbol{x}_m\|^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$$

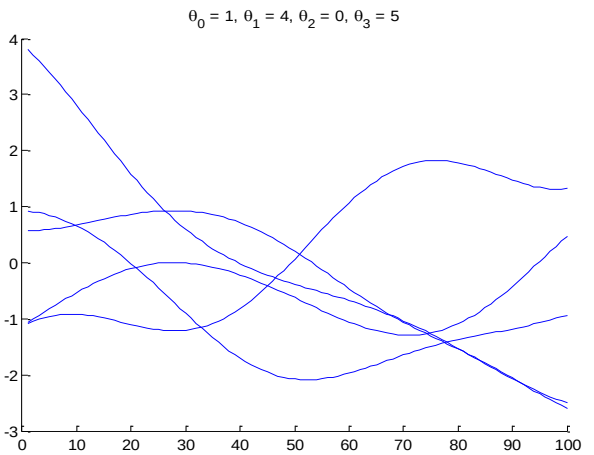












MatLab Code

Distribution Over Functions

- Sampling of data points $\{t_n\}$ from a Gaussian process.

- The blue curve shows a sample function from the Gaussian process prior over functions.

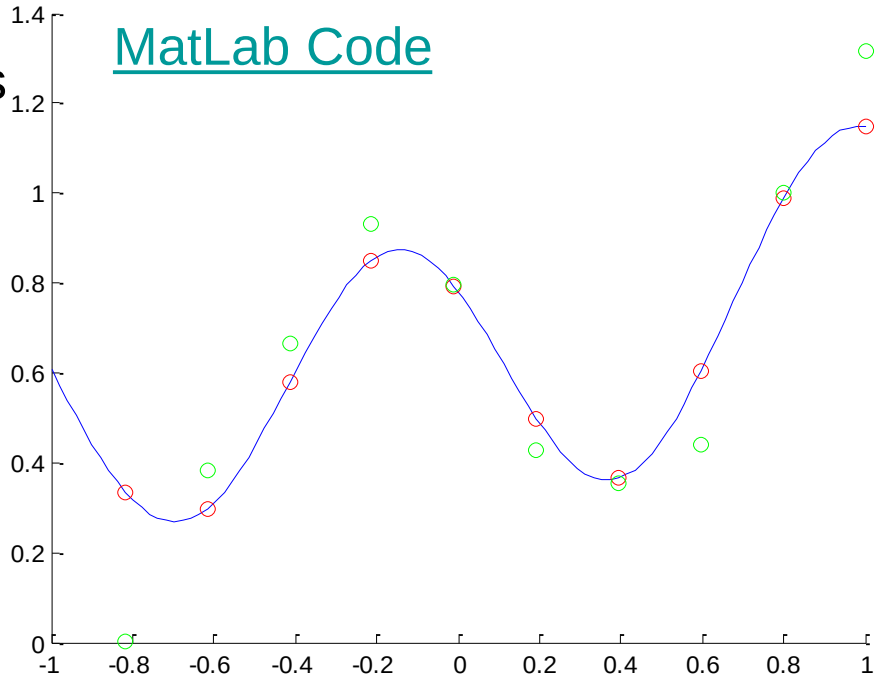
$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K})$$

- The red points show the values of y_n by evaluating the function at a set of input values $\{x_n\}$.

- The corresponding values of $\{t_n\}$, shown in green, are obtained by adding independent Gaussian noise to each of the $\{y_n\}$.

$$p(\boldsymbol{t}) = \mathcal{N}(\boldsymbol{t} \mid 0, \boldsymbol{C})$$

MatLab Code



Making Predictions

- So far we have a model for the joint probability distribution over sets of data points.
- For predictions t_{N+1} for a new input variable x_{N+1}, we need to evaluate the predictive distribution $p(t_{N+1}|\boldsymbol{t}_N)$.
- By partitioning the joint Gaussian distribution over $x_1, \ldots, x_N, x_{N+1}$, we obtain $p(t_{N+1}|\boldsymbol{t})$ given by its mean and covariance (see following Appendix with results for linear Gaussian models discussed in an earlier lecture).
- The joint distribution over $t_1, \ldots, t_N, t_{N+1}$ is:

$$p(\boldsymbol{t}_{N+1}) = \mathcal{N}(\boldsymbol{t}_{N+1} \,|\, \boldsymbol{0}, \boldsymbol{C}_{N+1})$$

- We partition the covariance matrix as follows:

Making Predictions

$$p\left(\boldsymbol{t}_{N+1}\right)=\mathcal{N}\left(\boldsymbol{t}_{N+1}\,|\,\boldsymbol{0},\boldsymbol{C}_{N+1}\right)$$

$$\boldsymbol{C}_{N+1}=\begin{pmatrix}\boldsymbol{C}_N & \boldsymbol{k}\\ \boldsymbol{k}^T & c\end{pmatrix}$$

- Here $\boldsymbol{C}_N$ is the $N\times N$ covariance matrix with elements given by

$$\boldsymbol{C}_N\left(\boldsymbol{x}_n,\boldsymbol{x}_m\right)=k\left(\boldsymbol{x}_n,\boldsymbol{x}_m\right)+\beta^{-1}\delta_{nm},\ n,m=1,\ldots,N$$

- The vector $\boldsymbol{k}$ has elements $k(\boldsymbol{x}_n,\boldsymbol{x}_{N+1})$ for $n=1,\ldots,N$, and the scalar

$$c=k\left(\boldsymbol{x}_{N+1},\boldsymbol{x}_{N+1}\right)+\beta^{-1}$$

Appendix: Conditional Gaussian Distributions

- If two sets of variables are jointly Gaussian, then the conditional distribution of one set conditioned on the other is again Gaussian.

- Suppose $\boldsymbol{x}$ is a $\boldsymbol{D}$-dimensional vector with Gaussian distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$ and that we partition $\boldsymbol{x}$ into two disjoint subsets $\boldsymbol{x_a}$ (M components) and $\boldsymbol{x_b}$ ($D-M$ components).

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_a \\ \boldsymbol{x}_b \end{pmatrix}$$

Appendix: Conditional Gaussian Distributions

- This partition also implies similar partitions for the mean and covariance.

$$\mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}$$

- $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$ implies that $\boldsymbol{\Sigma_{aa}}$ and $\boldsymbol{\Sigma_{bb}}$ are symmetric and

$$\Sigma_{ba} = \Sigma_{ab}^T$$

Appendix: The Precision Matrix

- We define the precision matrix $\boldsymbol{\Lambda}$ as Σ^{-1}.

- Its partition is given as above

$$\Lambda = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix}$$

where from $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$ we conclude that $\boldsymbol{\Lambda_{aa}}$ and $\boldsymbol{\Lambda_{bb}}$ are symmetric and

$$\Lambda_{ba} = \Lambda_{ab}^T$$

- Note that the above partition does NOT imply that $\boldsymbol{\Lambda_{aa}}$ is the inverse of $\boldsymbol{\Sigma_{aa}}$, etc.

Appendix: The Conditional Distribution

- We are now interested to compute $p(\boldsymbol{x}_a|\boldsymbol{x}_b)$. An easy way to do that is to look at $p(\boldsymbol{x}_a, \boldsymbol{x}_b)$ considering $\boldsymbol{x}_b$ constant.

- We have proved that the conditional is Gaussian with mean and variance given as:

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \Lambda_{aa}^{-1}\Lambda_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b) = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)$$

$$\Sigma_{a|b} = \Lambda_{aa}^{-1} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$$

- Note that the conditional mean is linear in $\boldsymbol{x}_b$ and the conditional variance is independent of $\boldsymbol{x}_b$.

Making Predictions

- Using the results from the Appendix, $p(t_{N+1}|\boldsymbol{t})$ is a Gaussian with mean and covariance:

$$\boldsymbol{t} = \left(t_1, \ldots, t_N, t_{N+1}\right)^T = (b \quad a)^T \qquad \boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{pmatrix} \qquad p\left(\boldsymbol{t}_{N+1}\right) = \mathcal{N}\left(\boldsymbol{t}_{N+1} \,|\, \boldsymbol{0}, \boldsymbol{C}_{N+1}\right)$$

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(x_b - \boldsymbol{\mu}_b) \Rightarrow \qquad \boldsymbol{\mu}_{t_{N+1}|\boldsymbol{t}_N} = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N$$

$$\boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} \Rightarrow \boldsymbol{\Sigma}_{t_{N+1}|\boldsymbol{t}_N} = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}$$

- Finally we obtain $p(t_{N+1}|\boldsymbol{t})$ with both the mean and variance functions of $\boldsymbol{x}_{n+1}$

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1}), \quad a_n = \left(\boldsymbol{C}_N^{-1} \boldsymbol{t}_N\right)_n,$$

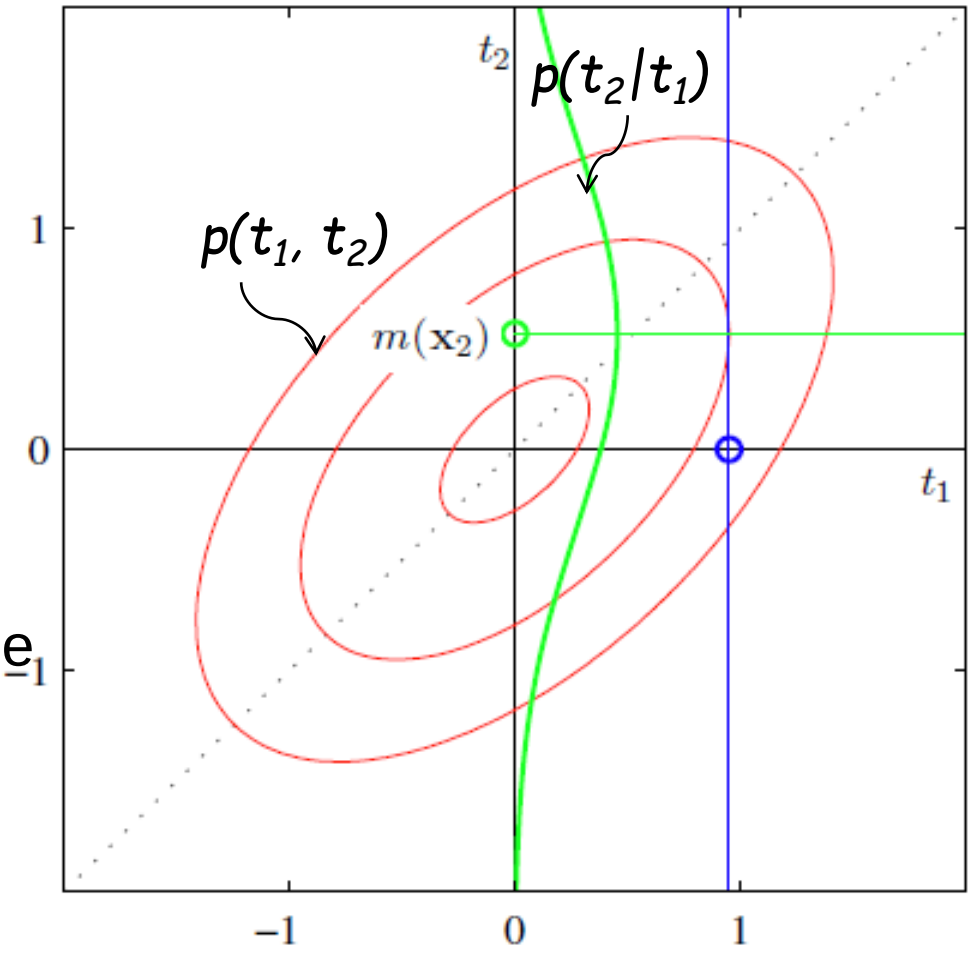
$$\boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

- If $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ depends only on $||\boldsymbol{x}_n - \boldsymbol{x}_m||$, then we obtain an expansion in radial basis functions.

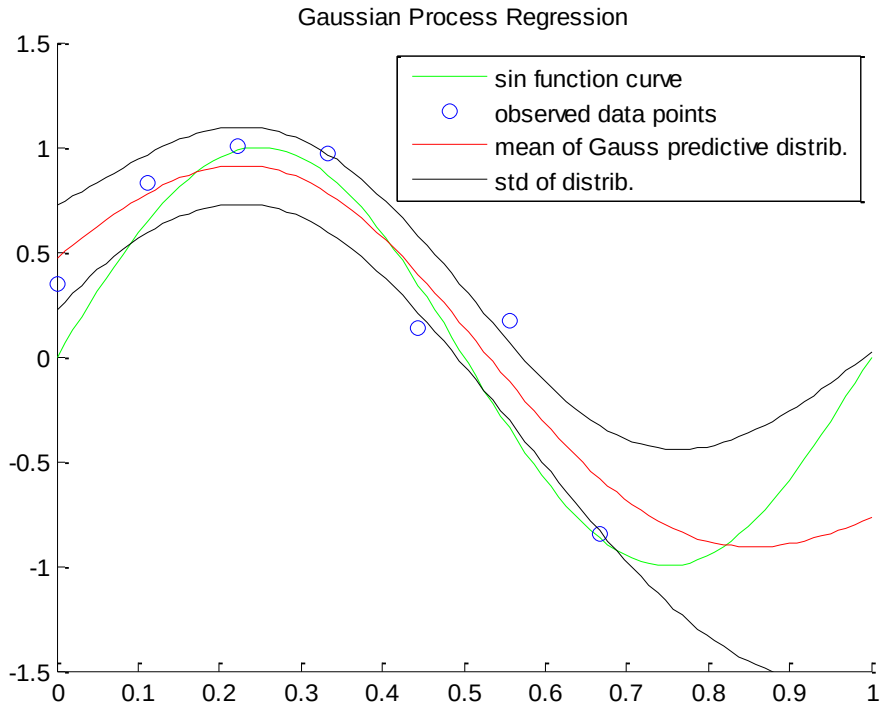
Gaussian Process Regression

- Illustration of the mechanism of Gaussian process regression for the case of one training point and one test point.
- The red ellipses show contours of the joint distribution $p(t_1, t_2)$.
- t_1 *is the* training data point, and conditioning on the value of t_1 *(*blue line), we obtain $p(t_2|t_1)$ shown as a function of t_2 by the green curve.



Making Predictions

- Green curve: original sinusoidal function; blue points: sampled training data points with additional noise; red line: mean estimate; black lines: $\pm 2\sigma$.
- Note the increase of uncertainty in the right of the data points.



MatLab Code

Restrictions on the kernel Matrix

- The covariance matrix must be positive definite.

$$C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}, n, m = 1, \ldots, N$$

- If λ_i is an eigenvalue of $\boldsymbol{K}$, then the corresponding eigenvalue of $\boldsymbol{C}$ will be $\lambda_i + \beta^{-1}$.

- It is therefore sufficient that the kernel matrix $k(\boldsymbol{x_n}, \boldsymbol{x_m})$ be positive semidefinite for any pair of points $\boldsymbol{x}_n$ and $\boldsymbol{x_m}$, so that $\lambda_i > 0$. Any $\lambda_i = 0$ will still give rise to a positive eigenvalue for $\boldsymbol{C}$ because $\beta > 0$.

- This is the same restriction on the kernel function discussed earlier, and so we can exploit the techniques discussed there.

Making Predictions

- Note that the mean of the conditional $p(t_{N+1}|\boldsymbol{t})$

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), ..., k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

can be written as:

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1}),$$

where

$$\boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \left(a_1 ... a_N\right)^T$$

- Thus if the kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ depends only on the distance $\|\boldsymbol{x}_n - \boldsymbol{x}_m\|$ we obtain an expansion in radial basis.

Making Predictions

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), ..., k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

- These results define the predictive distribution for Gaussian process regression with an arbitrary kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$
- If the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ is defined in terms of a finite set of basis functions, the results become identical to those we discussed in an earlier lectures on regression (see proof in the next two slides)

$$p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) = \int p(t \mid x, \boldsymbol{w})\, p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{t}) d\boldsymbol{w} = \mathcal{N}\left(t \mid \boldsymbol{m}_N^T \boldsymbol{\phi}(x), \sigma_N^2(x)\right)$$

$$\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}$$

$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

$$\sigma_N^2(x) = \frac{1}{\beta} + \boldsymbol{\phi}(x)^T S_N \boldsymbol{\phi}(x)$$

Kernels Defined by a Finite Basis

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

- We can write the vector $\boldsymbol{k}$ as:

$$\boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T = \frac{1}{\alpha} \begin{pmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\boldsymbol{x}_N) \end{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) = \frac{1}{\alpha} \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})$$

and use

$$\boldsymbol{C}_N = \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N$$

- Using this, the predicted mean takes the form

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \frac{1}{\alpha} \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \boldsymbol{\Phi}^T \left(\alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N\right)^{-1} \boldsymbol{t}_N =$$

$$\alpha^{-1} \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \underset{\boldsymbol{A}}{\beta \boldsymbol{\Phi}^T} \left(\underset{\boldsymbol{B}}{\alpha^{-1} \boldsymbol{\Phi}} \underset{\boldsymbol{A}}{\beta \boldsymbol{\Phi}^T} + \boldsymbol{I}_N\right)^{-1} \boldsymbol{t}_N \underset{(\boldsymbol{I}+\boldsymbol{AB})^{-1}\boldsymbol{A}=\boldsymbol{A}(\boldsymbol{I}+\boldsymbol{BA})^{-1}}{=} \alpha^{-1} \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \left(\beta \boldsymbol{\Phi}^T \alpha^{-1} \boldsymbol{\Phi} + \boldsymbol{I}_M\right)^{-1} \beta \boldsymbol{\Phi}^T \boldsymbol{t}_N =$$

$$\boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \beta \left(\beta \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \alpha \boldsymbol{I}_M\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}_N = \beta \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}_N, \text{ the required result}$$

Kernels Defined by a Finite Basis

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

- Using $\boldsymbol{k} = \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})$, $c = \alpha^{-1} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) + \beta^{-1}$ and $\boldsymbol{C}_N = \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N$, we compute the predictive variance as:

$$\sigma^2(\boldsymbol{x}_{N+1}) = \alpha^{-1} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) + \beta^{-1} - \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \boldsymbol{\Phi}^T \alpha^{-1} \left(\alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N \right)^{-1} \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) =$$

$$\beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \left\{ \underbrace{\alpha^{-1} \boldsymbol{I}_M}_{\boldsymbol{A}^{-1}} - \underbrace{\boldsymbol{\Phi}^T}_{\boldsymbol{B}} \underbrace{\alpha^{-1} \boldsymbol{I}_M}_{\boldsymbol{A}^{-1}} \left(\underbrace{\alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T}_{\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}} + \underbrace{\beta^{-1} \boldsymbol{I}_N}_{\boldsymbol{D}} \right)^{-1} \underbrace{\boldsymbol{\Phi}}_{\boldsymbol{C}} \underbrace{\alpha^{-1} \boldsymbol{I}_M}_{\boldsymbol{A}^{-1}} \right\} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) =$$

$\beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T (\alpha \boldsymbol{I}_M + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) = \beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T S_N \boldsymbol{\phi}(\boldsymbol{x}_{N+1})$, the required result

- Here we used the Woodbury identity with ***A***, ***B***, ***C***, ***D*** as shown.

$$(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1}$$

Gaussian process Vs. Basis Function Model

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

$$p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) = \mathcal{N}\left(t \mid \boldsymbol{m}_N^T \boldsymbol{\phi}(x), \sigma_N^2(x)\right)$$

$$\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}$$
$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

$$\sigma_N^2(x) = \frac{1}{\beta} + \boldsymbol{\phi}(x)^T S_N \boldsymbol{\phi}(x)$$

- In Gaussian processes, we invert $\boldsymbol{C}_N$ of size $N \times N$, cost $\mathcal{O}(\mathrm{N}^3)$
- In the basis function model, we invert S_N, size $\mathrm{M} \times M$, cost $\mathcal{O}(\mathrm{M}^3)$
- These matrix inversions must be performed once for the given training set.
- For each new test point, both methods require a vector-matrix multiplications ($\mathcal{O}(\mathrm{N}^2)$ vs. $\mathcal{O}(\mathrm{M}^2)$). If M < N, it will be computationally more efficient to work in the basis function framework.
- An advantage of Gaussian processes is that we can consider covariance functions that can only be expressed in terms of an infinite number of basis functions.

Large Training Data Sets

- For large training data sets, GPs can become infeasible, and approximation schemes have been developed with better scaling with training set size than the exact approach.

- Gibbs, M. N. (1997). *Bayesian Gaussian processes for regression and classification*. Phd thesis, University of Cambridge.
- Tresp, V. (2001). Scaling kernel-based systems to large data sets. *Data Mining and Knowledge Discovery* **5**(3), 197–211.
- Smola, A. J. and P. Bartlett (2001). Sparse greedy Gaussian process regression. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13, pp. 619–625. MIT Press.
- Williams, C. K. I. and M. Seeger (2001). Using the Nystrom method to speed up kernel machines. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13, pp. 682–688. MIT Press.
- Csató, L. and M. Opper (2002). Sparse on-line Gaussian processes. *Neural Computation* **14**(3), 641– 668.
- Seeger, M., C. K. I. Williams, and N. Lawrence (2003). Fast forward selection to speed up sparse Gaussian processes. In C. M. Bishop and B. Frey (Eds.), *Proceedings Ninth International Workshop on Artificial Intelligence and Statistics, Key West, Florida*.
- Bishop, C. M. and I. T. Nabney (2008). *Pattern Recognition and Machine Learning: A Matlab Companion*. Springer. In preparation.

Extension of Gaussian Processes

- Extension of GPs to multiple target variables (co-kriging) is straightforward.
- Other extensions of GPs include
  - modelling the distribution over low-dimensional manifolds for unsupervised learning and
  - the solution of SPDEs

- Cressie, N. (1993). *Statistics for Spatial Data*.Wiley.
- Bishop, C. M., M. Svensén, and C. K. I. Williams (1998a). Developments of the Generative Topographic Mapping. *Neurocomputing* **21**, 203– 224.
- Graepel, T. (2003). Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 234–241.

Learning Hyperparameters θ

- Instead of fixing the covariance function, use a parametric family of functions and infer the parameters from the data.
- Parameters $\boldsymbol{\theta}$: length scale of correlations, and precision of noise β.
- Techniques for learning $\boldsymbol{\theta}$ are based on the likelihood $p(\boldsymbol{t}|\boldsymbol{\theta})$ – analogous to model evidence techniques used earlier.
- Simplest approach:
  - Maximizing the log-likelihood (MLE): $\ln p(\boldsymbol{t}|\boldsymbol{\theta})$

$$\ln p(\boldsymbol{t}\,|\,\boldsymbol{\theta}) = -\frac{1}{2}\ln\left|\boldsymbol{C}_N\right| - \frac{1}{2}\boldsymbol{t}^T\boldsymbol{C}_N^{-1}\boldsymbol{t} - \frac{N}{2}\ln(2\pi)$$

  - Use gradient optimization

$$\frac{\partial}{\partial\theta_i}\ln p(\boldsymbol{t}\,|\,\boldsymbol{\theta}) = -\frac{1}{2}tr\left[\boldsymbol{C}_N^{-1}\frac{\partial\boldsymbol{C}_N}{\partial\theta_i}\right] + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{C}_N^{-1}\frac{\partial\boldsymbol{C}_N}{\partial\theta_i}\boldsymbol{C}_N^{-1}\boldsymbol{t}$$

  - $\ln p(\boldsymbol{t}|\,\boldsymbol{\theta})$ is in general non-convex and can have multiple maxima

- Fletcher, R. (1987). *Practical Methods of Optimization* (Second ed.). Wiley.
- Nocedal, J. and S. J. Wright (1999). *Numerical Optimization*. Springer.

Appendix

- Show that

$$\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1}$$

Indeed note that

$$\frac{\partial}{\partial x}\left(\boldsymbol{A}\boldsymbol{A}^{-1}\right) = \boldsymbol{0} \Rightarrow \frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1} + \boldsymbol{A}\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = 0 \Rightarrow \frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1}$$

- Show that ($\boldsymbol{A}$ symmetric)

$$\frac{\partial}{\partial x}\ln|\boldsymbol{A}| = Tr\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\right)$$

Indeed using the eigen-decomposition of $\boldsymbol{A}$, note that

$$\boldsymbol{A} = \sum_i \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^T,\ \boldsymbol{A}^{-1} = \sum_i \frac{1}{\lambda_i}\boldsymbol{u}_i \boldsymbol{u}_i^T,\ tr\boldsymbol{A} = \sum_i \lambda_i,\ |\boldsymbol{A}| = \prod_i \lambda_i \Rightarrow$$

$$\ln|\boldsymbol{A}| = \sum_i \ln\lambda_i \Rightarrow \frac{\partial}{\partial x}\ln|\boldsymbol{A}| = \sum_i \frac{\partial \ln\lambda_i}{\partial x} = \sum_i \frac{1}{\lambda_i}\frac{\partial \lambda_i}{\partial x} = Tr\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\right)$$

Learning Hyperparameters θ

- One can also introduce a prior $p(\boldsymbol{\theta})$ and maximize the log-posterior (maximum a posteriori): $\ln p(\boldsymbol{t}|\,\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})$
- In a Bayesian setting, we need to compute marginals over $\boldsymbol{\theta}$ weighted with $p(\boldsymbol{t}|\,\boldsymbol{\theta})p(\boldsymbol{\theta})$; this is not tractable → use approximations
- We have assumed that the contribution to the predictive variance arising from the noise β is constant. The noise might not be additive but dependent on x (heteroscedastic)

$$C\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) + \beta^{-1}\delta_{nm}, \; n, m = 1, \ldots, N$$

- A second Gaussian process can be introduced to represent the dependency of β on $\boldsymbol{x}$.
  - Use a Gaussian process for $\ln \beta$ since $\beta \geq 0$.

- Goldberg, P. W., C. K. I. Williams, and C. M. Bishop (1998). Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems*, Volume 10, pp. 493–499. MIT Press.

Automatic Relevance Detection

- An additional hyperparameter can be introduced for each input dimension, e.g. in two-dimensions:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\left(-\frac{1}{2} \sum_{i=1}^{2} \eta_i \left(x_i - x_i^{'} \right)^2 \right)$$

- Hyperparameter optimization by maximum likelihood allows then a different weighting of each dimension.

- Unrelevant dimensions (with small weights) can be detected and discarded.

Automatic Relevance Detection

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\left(-\frac{1}{2} \sum_{i=1}^{2} \eta_i \left(x_i - x_i^{'} \right)^2 \right)$$

- Samples from the resulting prior over functions $y(\boldsymbol{x})$ are shown next for two settings of η_i.
- As a particular η_i becomes small, the function becomes insensitive to x_i.
- By adapting these parameters to a data set using MLE, it becomes possible to detect input variables that have little effect on the predictive distribution (such inputs are discarded)

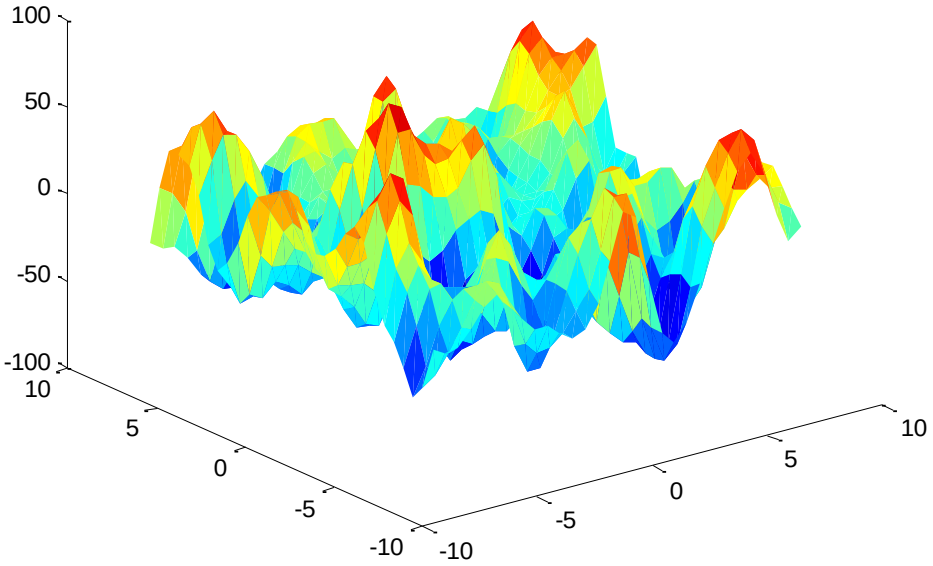
- MacKay, D. J. C. (1994). Bayesian methods for backprop networks. In E. Domany, J. L. van Hemmen, and K. Schulten (Eds.), *Models of Neural Networks, III*, Chapter 6, pp. 211–254. Springer.
- Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer. Lecture Notes in Statistics 118.

Automatic Relevance Detection

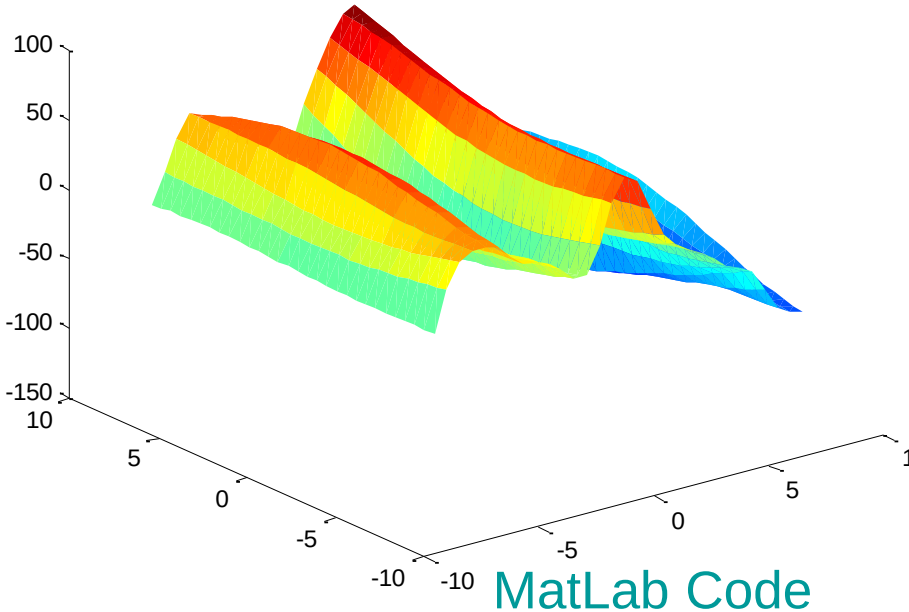
$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\left(-\frac{1}{2} \sum_{i=1}^{2} \eta_i \left(x_i - x_i^{'} \right)^2 \right)$$

- Samples from the resulting prior over functions $y(\boldsymbol{x})$ are shown for two settings of η_i.

$\eta_1 = \eta_2 = 1$



$\eta_1 = 1, \eta_2 = 0.01$



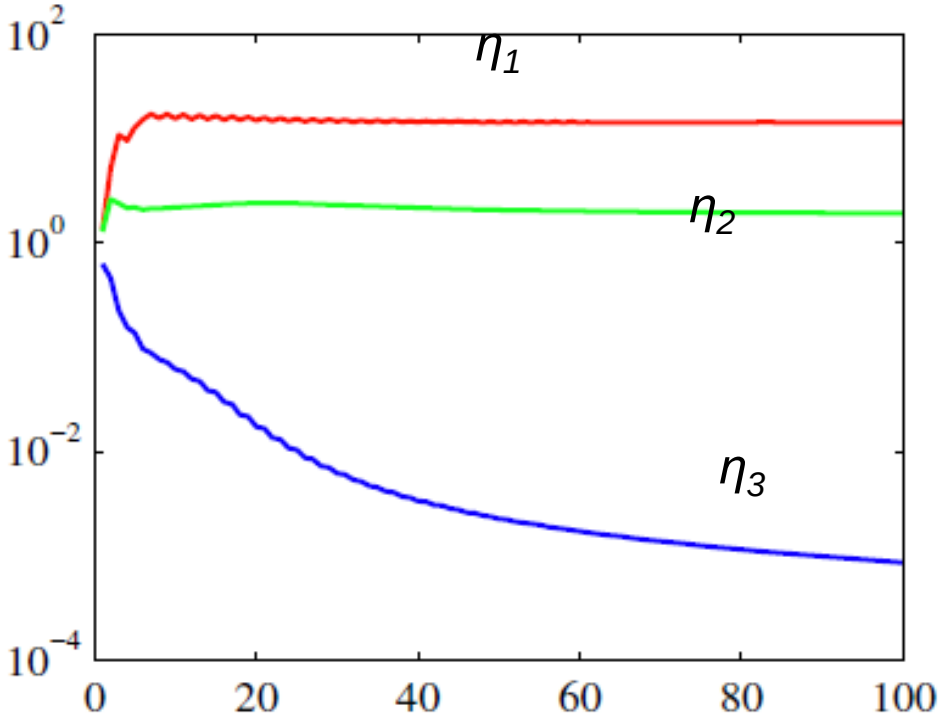
MatLab Code

Automatic Relevance Detection

- ARD is illustrated next using a data set having 3 inputs x_1, x_2 and x_3.
- The target variable t, is generated by sampling 100 values of x_1 from a Gaussian, evaluating the function $\sin(2\pi x_1)$, and then adding Gaussian noise.
- Values of x_2 are given by copying the corresponding values of x_1 and adding noise, and values of x_3 are sampled from an independent Gaussian distribution.
- Thus x_1 is a good predictor of t, x_2 is a more noisy predictor of t, and x_3 has only chance correlating with t.
- The marginal likelihood for a Gaussian process with ARD parameters η_1, η_2, η_3 is optimized using the scaled conjugate gradients algorithm.

Automatic Relevance Detection

- We see (log scale in the vertical axis) that η_1 converges to a relatively large value, η_2 converges to a much smaller value, and η_3 becomes very small indicating that x_3 is irrelevant for predicting t.



MatLab Code

- Nabney, I. T. (2002). *Netlab: Algorithms for Pattern Recognition*. Springer.

Automatic Relevance Detection

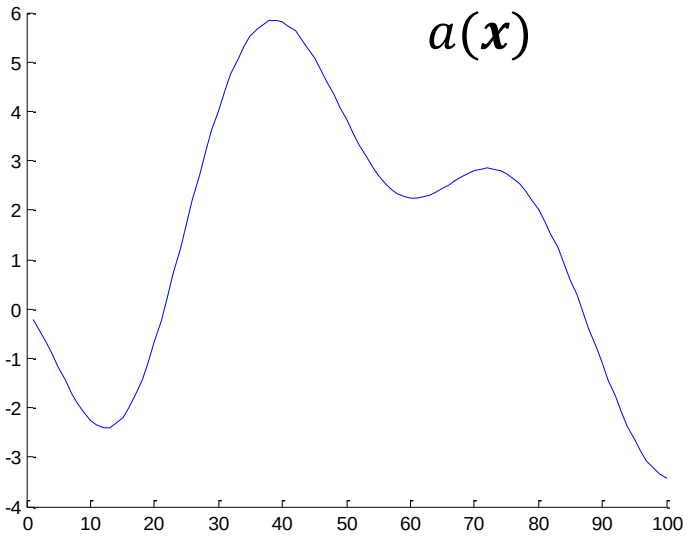
- The ARD framework can be applied to general kernels of the form:

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \sum_{i=1}^{D} \eta_i \left(x_{ni} - x_{mi} \right)^2 \right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$$

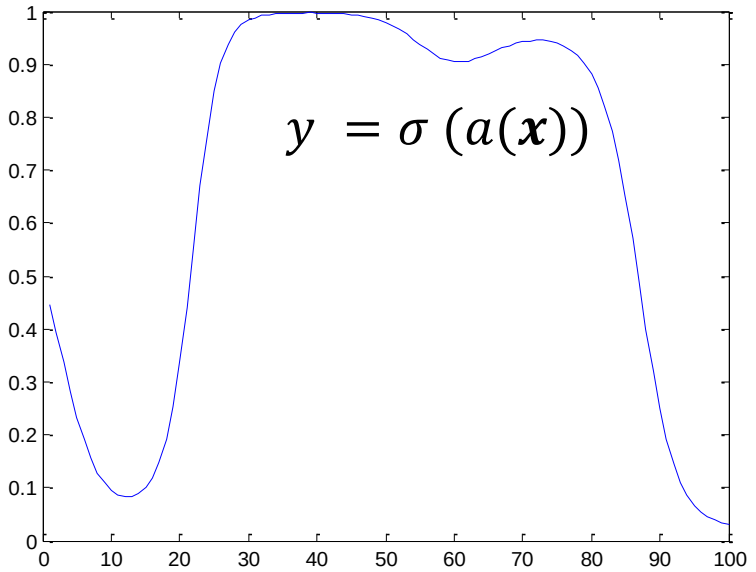
where D is the input dimensionality.

Gaussian Processes for Classification

- Objective: model posterior probabilities of the target variable for a new input
- Problem: we need to map values to interval $(0, 1)$
- Solution: use a Gaussian process together with a non-linear activation function (e.g., sigmoid)



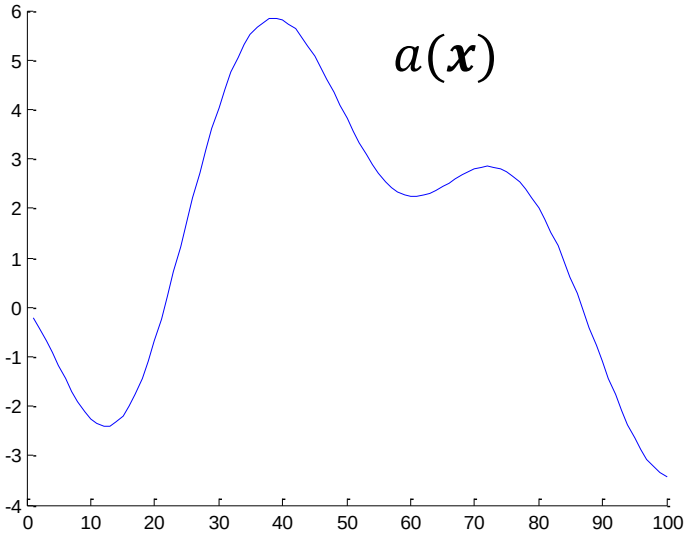
Sample from a Gaussian process over $a(x)$

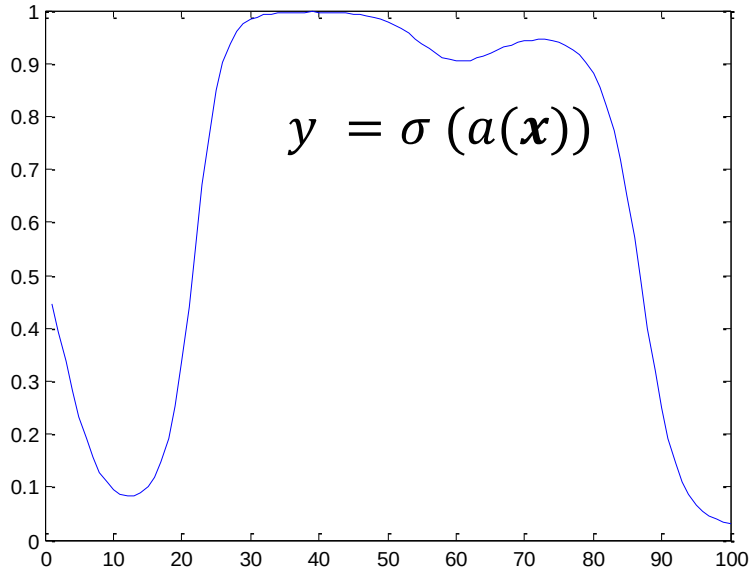


MatLab Code

Gaussian Process for Classification

- Consider a two-class problem with target values t ∈ {0, 1}. Define a Gaussian process over a function $a(\boldsymbol{x})$ and transform a using the logistic sigmoid to $y = \sigma(a(\boldsymbol{x}))$.
- This will give a non-Gaussian stochastic process over functions $y(\boldsymbol{x})$ where y ∈ {0, 1}.





MatLab Code

- The probability over the target t is a Bernoulli: $p(t \mid a) = \sigma(a)^t \left(1 - \sigma(a)\right)^{1-t}$

Gaussian Process for Classification

- We denote the training set inputs by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with corresponding observed target variables $\boldsymbol{t} = (t_1, \ldots, t_N)^T$.
- We also consider a single test point $\boldsymbol{x}_{N+1}$ with target value t_{N+1}.
- Our goal is to determine the predictive distribution $p(t_{N+1}|\boldsymbol{t})$, where we left the conditioning on the input variables implicit.
- Introduce a Gaussian process prior over the vector $\boldsymbol{a}_{N+1}$, which has components $a(\boldsymbol{x}_1), \ldots, a(x_{N+1})$.
- This defines a non-Gaussian process over $\boldsymbol{t}_{N+1}$, and by conditioning on $\boldsymbol{t}_N$ we obtain the required predictive distribution $p(t_{N+1}|\boldsymbol{t})$.
- The Gaussian process prior for $\boldsymbol{a}_{N+1}$ takes the form

$$p(\boldsymbol{a}_{N+1}) = \mathcal{N}(\boldsymbol{a}_{N+1} \mid 0, \boldsymbol{C}_{N+1})$$

Gaussian Process for Classification

- The covariance matrix does not include a noise term because we assume that all training data points are correctly labeled.

- However, for numerical reasons it is convenient to introduce a noise-like term governed by a parameter ν that ensures that the covariance matrix is positive definite. Thus the covariance matrix $\boldsymbol{C}_{N+1}$ has elements given by

$$\boldsymbol{C}\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) + \nu\delta_{nm}$$

- The kernel k is positive semidefinite and v is fixed. $k(\boldsymbol{x}, \boldsymbol{x}')$ is governed by parameters θ that we will compute from the training data.

- For two class problems, it is sufficient to predict:

$$p(t_{N+1} = 1 \mid \boldsymbol{t}_N)$$

Gaussian Process for Classification

- The predictive distribution is given as:

$$p(t_{N+1}=1|\boldsymbol{t}_N)=\int p(t_{N+1}=1,a_{N+1}|\boldsymbol{t}_N)\,da_{N+1}$$

$$=\int p(t_{N+1}=1|a_{N+1})\,p(a_{N+1}|\boldsymbol{t}_N)\,da_{N+1}=\int \sigma(a_{N+1})\,p(a_{N+1}|\boldsymbol{t}_N)\,da_{N+1}$$

- The integral is analytically intractable. Can be approximated with sampling (Neal)
- Integration can be analytical if $p(a_{N+1}|t_N)$ can be approximated with a Gaussian (Williams & Barber, Gibbs & MacKay, Gibbs)
- Approximations based on EP can also be used (Opper & Winther, Minka, Seeger)

- Neal, R. M. (1997). Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report 9702, Department of Computer Statistics, University of Toronto.
- Williams, C. K. I. and D. Barber (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**, 1342–1351.
- Gibbs, M. N. and D. J. C. MacKay (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks* **11**, 1458–1464.
- Gibbs, M. N. (1997). *Bayesian Gaussian processes for regression and classification*. Phd thesis, University of Cambridge.
- Opper, M. and O. Winther (2000b). Gaussian processes for classification. *Neural Computation* **12**(11), 2655–2684.
- Minka, T. (2001b). *A family of approximate algorithms for Bayesian inference*. Ph. D. thesis, MIT.
- Seeger, M. (2003). *Bayesian Gaussian Process Models: PAC-Bayesian Generalization Error Bounds and Sparse Approximations*. Ph. D. thesis, University of Edinburg.

Gaussian Process for Classification

- We need to compute $p(t_{N+1}=1|\boldsymbol{t}_N)=\int \sigma(a_{N+1})\, p(a_{N+1}|\boldsymbol{t}_N)\, da_{N+1}$
- One can compute the convolution of a Gaussian and a sigmoid function:

$$\int \sigma(a)\mathcal{N}(a|\mu,\sigma^2)da \simeq \sigma\left(\kappa(\sigma^2)\mu\right)$$

$$\kappa(\sigma^2)=\sqrt{1+\frac{\pi\sigma^2}{8}}$$

We can thus try to approximate the posterior distribution $p(a_{N+1}|\boldsymbol{t}_N)$ as Gaussian

- How do we approximate the posterior?
  - Variational inference and make use of the local variational bound on the logistic sigmoid.
  - Expectation propagation
  - Laplace approximation

Laplace Approximation

- We can rewrite the posterior over a_{N+1} using Bayes' theorem:

$$
\begin{aligned}
p(a_{N+1} \mid \boldsymbol{t}_N) &= \int p(a_{N+1}, \boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N \\
&= \frac{1}{p(\boldsymbol{t}_N)} \int p(a_{N+1}, \boldsymbol{a}_N)\, p(\boldsymbol{t}_N \mid a_{N+1}, \boldsymbol{a}_N)\, d\boldsymbol{a}_N \\
&= \frac{1}{p(\boldsymbol{t}_N)} \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N)\, p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)\, d\boldsymbol{a}_N \\
&= \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N
\end{aligned}
$$

- We know how to compute mean and covariance for $p(a_N|\boldsymbol{a}_N)$.

$$
p(a_{N+1} \mid \boldsymbol{a}_N) = \mathcal{N}\left(\boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N,\, c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}\right)
$$

- It remains to find a Gaussian approximation only for $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$

- This is done noting that $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N)$

Laplace Approximation

- We rewrote the posterior over a_{N+1} using Bayes' theorem:

$$p(a_{N+1} \mid \boldsymbol{t}_N) = \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N$$

where

$$p(a_{N+1} \mid \boldsymbol{a}_N) = \mathcal{N}\left(\boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N, c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}\right)$$

- Also using $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) p(\boldsymbol{a}_N)$, $p(\boldsymbol{a}_N) = \mathcal{N}\left(0, \boldsymbol{C}_N\right)$ and

$$p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) = \prod_{n=1}^{N} \sigma(a_n)^{t_n} \left(1 - \sigma(a_n)\right)^{1-t_n} = \prod_{n=1}^{N} e^{a_n t_n} \sigma(-a_n)$$

where in the last derivation we used: $\sigma(a_n) = \dfrac{1}{1+e^{-a_n}}$ and

$$\sigma(a_n)^{t_n} \left(1 - \sigma(a_n)\right)^{1-t_n} = \underbrace{\left(1 - \sigma(a_n)\right)}_{\sigma(-a_n)} \left(\underbrace{\frac{\sigma(a_n)}{1 - \sigma(a_n)}}_{e^{a_n}} \right)^{t_n} = e^{a_n t_n} \sigma(-a_n)$$

Laplace Approximation

- The log of $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) p(\boldsymbol{a}_N)$ which up to an additive constant is:

$$p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{a}_N) p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)$$

$$\propto \mathcal{N}(0, \boldsymbol{C}_N) \prod_{n=1}^{N} e^{a_n t_n} \underbrace{\sigma(-a_n)}_{1/(1+e^{a_n})} \Rightarrow$$

$$\Psi(\boldsymbol{a}_N) = \ln p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) = \ln p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) + \ln p(\boldsymbol{a}_N)$$

$$= -\frac{1}{2}\boldsymbol{a}_N^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T \boldsymbol{a}_N - \sum_{n=1}^{N} \ln\left(1 + e^{a_n}\right)$$

- We will need to compute the 2nd derivative of this to come up with the Laplace approximation.

Laplace Approximation

$$\Psi(\boldsymbol{a}_N) = -\frac{1}{2}\boldsymbol{a}_N^T\boldsymbol{C}_N^{-1}\boldsymbol{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T\boldsymbol{a}_N - \sum_{n=1}^{N}\ln\left(1+e^{a_n}\right)$$

- The gradients are given as

$$\nabla\Psi(\boldsymbol{a}_N) = \boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N, \quad \boldsymbol{\sigma}_N = \left(\sigma(a_1),...,\sigma(a_N)\right)^T$$

$$\nabla^2\Psi(\boldsymbol{a}_N) = -\boldsymbol{W}_N - \boldsymbol{C}_N^{-1}, \quad \boldsymbol{W}_N = diag\left(\sigma(a_1)(1-\sigma(a_1)),...,\sigma(a_N)(1-\sigma(a_N))\right)$$

- We cannot find the mode by setting $\nabla\Psi(\boldsymbol{a}_N) = 0$ since $\boldsymbol{\sigma}_N$ depends nonlinearly on $\boldsymbol{a}_N$. For this we will use the IRLS algorithm.
- Note that $0 < \sigma(a_1)(1-\sigma(a_1)) < 1/4 \Rightarrow \boldsymbol{W}_N$ is positive definite.
- The Hessian $A = -\nabla^2\Psi(\boldsymbol{a}_N)$ is positive definite and thus the posterior $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$ is log convex and has a single global mode.
- At the mode, the gradient vanishes giving: $\boldsymbol{a}_N^* = \boldsymbol{C}_N(\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$

Laplace Approximation

- We can use the iterative reweighted least squares (IRLS) algorithm ($\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$) to update for $\boldsymbol{a}_N$ (find the minimum of $-\ln p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)$).

- The iterative update equation for $\boldsymbol{a}_N$ is:

$$\boldsymbol{a}_N^{new} = \boldsymbol{C}_N(\boldsymbol{I} + \boldsymbol{W}_N\boldsymbol{C}_N)^{-1}\left(\boldsymbol{t}_n - \boldsymbol{\sigma}_n + \boldsymbol{W}_N\boldsymbol{a}_N\right)$$

Here we used

$$\begin{aligned}
\boldsymbol{a}_N^{new} &= \boldsymbol{a}_N + (\boldsymbol{W}_N + \boldsymbol{C}_N^{-1})^{-1}\left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N\right) = \boldsymbol{a}_N + \boldsymbol{C}_N(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})^{-1}\left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N\right) \\
&= \boldsymbol{C}_N(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})^{-1}\left\{(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})\boldsymbol{C}_N^{-1}\boldsymbol{a}_N + \left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N\right)\right\} \\
&= \boldsymbol{C}_N(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})^{-1}\left\{\boldsymbol{W}_N\boldsymbol{a}_N + \boldsymbol{t}_N - \boldsymbol{\sigma}_N\right\}
\end{aligned}$$

Laplace Approximation

- We can use the iterative reweighted least squares (IRLS) algorithm ($\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$) to update for $\boldsymbol{a}_N$.

- The iterative update equation for $\boldsymbol{a}_N$ is:

$$\boldsymbol{a}_N^{new} = \boldsymbol{C}_N(\boldsymbol{I} + \boldsymbol{W}_N\boldsymbol{C}_N)^{-1}\left(\boldsymbol{t}_n - \boldsymbol{\sigma}_n + \boldsymbol{W}_N\boldsymbol{a}_N\right)$$

- The mode position $\boldsymbol{a}_N^*$ and the Hessian matrix $\boldsymbol{H}$ at this position $\boldsymbol{H} = -\nabla^2\Psi(\boldsymbol{a}_N) = \boldsymbol{W}_N + \boldsymbol{C}_N^{-1}$ define the Gaussian approximation to $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)$

$$q(\boldsymbol{a}_N) = \mathcal{N}(\boldsymbol{a}_N \mid \boldsymbol{a}_N^*, \boldsymbol{H}^{-1})$$

In the Hessian note that $\boldsymbol{W}_N$ is computed using $\boldsymbol{a}_N^*$.

Laplace Approximation

- Now we can go back to the formulas for linear Gaussian models and compute the integrals

$$p(a_{N+1}\,|\,\boldsymbol{t}_N)=\int p(a_{N+1}\,|\,\boldsymbol{a}_N)\,p(\boldsymbol{a}_N\,|\,\boldsymbol{t}_N)\,d\boldsymbol{a}_N$$

$$p(a_{N+1}\,|\,\boldsymbol{a}_N)=\mathcal{N}\left(\boldsymbol{k}^T\boldsymbol{C}_N^{-1}\boldsymbol{a}_N,\,c-\boldsymbol{k}^T\boldsymbol{C}_N^{-1}\boldsymbol{k}\right)$$

$$p(\boldsymbol{a}_N\,|\,\boldsymbol{t}_N)\approx q(\boldsymbol{a}_N)=\mathcal{N}(\boldsymbol{a}_N\,|\,\boldsymbol{a}_N^*,\boldsymbol{H}^{-1}),\;\boldsymbol{H}=\boldsymbol{W}_N+\boldsymbol{C}_N^{-1}$$

from which we have:

$$p(a_{N+1}\,|\,\boldsymbol{t}_N)=\mathcal{N}\left(\boldsymbol{k}^T(\boldsymbol{t}_N-\boldsymbol{\sigma}_N),\,c-\boldsymbol{k}^T\left(\boldsymbol{W}_N^{-1}+\boldsymbol{C}_N\right)^{-1}\boldsymbol{k}\right)$$

- Finally, we can also compute $p(t_{N+1}|\boldsymbol{t})$ (or the decision boundary $p(t_{N+1}|\boldsymbol{t})=0.5$ – in this case $\mu=\boldsymbol{k}^T(\boldsymbol{t}_N-\boldsymbol{\sigma}_N)=0$)

$$p(t_{N+1}=1\,|\,\boldsymbol{t}_N)=\int\sigma(a_{N+1})\,p(a_{N+1}\,|\,\boldsymbol{t}_N)\,da_{N+1}$$

using $$\int\sigma(a)\mathcal{N}(a\,|\,\mu,\sigma^2)da\approx\sigma\left(\kappa(\sigma^2)\mu\right),\quad\kappa(\sigma^2)=\sqrt{1+\frac{\pi\sigma^2}{8}}$$

Learning the Hyperparameters

- To determine the parameters $\boldsymbol{\theta}$ of the covariance function, we can maximize the likelihood function:

$$p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \int p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{\theta}) d\boldsymbol{a}_N = \int p(\boldsymbol{t}_N, \boldsymbol{a}_N \mid \boldsymbol{\theta}) d\boldsymbol{a}_N$$

- The integral is analytically intractable. The Laplace approximation can be applied again: $\ln p(\boldsymbol{t}_N, \boldsymbol{a}_N) = \Psi(\boldsymbol{a}_N) = -\frac{1}{2}\boldsymbol{a}_N^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T \boldsymbol{a}_N - \sum_{n=1}^{N} \ln\left(1 + e^{a_n}\right)$

$$\ln p(\boldsymbol{t}_N, \boldsymbol{a}_N) \approx \Psi(\boldsymbol{a}_N^*) - \frac{1}{2}(\boldsymbol{a}_N - \boldsymbol{a}_N^*)^T (W_N + C_N^{-1})(\boldsymbol{a}_N - \boldsymbol{a}_N^*)$$

- Integrating in $\boldsymbol{a}_N$ and using the normalization factor of a Gaussian gives:

$$\ln p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \boldsymbol{\Psi}(\boldsymbol{a}_N^*) - \frac{1}{2}\ln\left|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right| + \frac{N}{2}\ln(2\pi)$$

where

$$\boldsymbol{\Psi}(\boldsymbol{a}_N^*) = \ln p(\boldsymbol{a}_N^* \mid \boldsymbol{\theta}) + \ln p(\boldsymbol{t}_N \mid \boldsymbol{a}_N^*)$$

- We now need an expression for the gradient of the log of $p(\boldsymbol{t}_N|\boldsymbol{\theta})$. Both $\boldsymbol{a}_N^*$ and $\boldsymbol{C}_N$ depend on θ

Learning the Hyperparameters

- To determine the parameters , maximize the likelihood:

$$\Psi(\boldsymbol{a}_N^*) = -\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T\boldsymbol{a}_N^* - \sum_{n=1}^{N}\ln\left(1+e^{a_n^*}\right)$$

$$\ln p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \Psi\left(\boldsymbol{a}_N^*\right) - \frac{1}{2}\ln|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}| + \frac{N}{2}\ln(2\pi)$$

- Using $\frac{\partial}{\partial x}\ln|\boldsymbol{A}| = Tr\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\right)$ and $\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1}$, we obtain

the following terms of $\frac{\partial}{\partial \theta_j}\ln p(\boldsymbol{t}_N \mid \theta)$ with explicit dependence on θ:

$$\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}Tr\left[\boldsymbol{C}_N^{-1}\left[\boldsymbol{I} - \left(\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right)^{-1}\boldsymbol{C}_N^{-1}\right]\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right] =$$

$$\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}Tr\left[\boldsymbol{C}_N^{-1}\left[\left(\boldsymbol{C}_N\boldsymbol{W}_N + \boldsymbol{I}\right)\left(\boldsymbol{C}_N\boldsymbol{W}_N + \boldsymbol{I}\right)^{-1} - \left(\boldsymbol{C}_N\boldsymbol{W}_N + \boldsymbol{I}\right)^{-1}\right]\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right]$$

$$\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}Tr\left[\left(I + \boldsymbol{C}_N\boldsymbol{W}_N\right)^{-1}\boldsymbol{W}_N\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right]$$

Learning the Hyperparameters

- To determine the parameters, maximize the likelihood:

$$\Psi(\boldsymbol{a}_N^*) = -\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T\boldsymbol{a}_N^* - \sum_{n=1}^{N}\ln\left(1+e^{a_n^*}\right)$$

$$\ln p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \Psi\left(\boldsymbol{a}_N^*\right) - \frac{1}{2}\ln\left|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right| + \frac{N}{2}\ln(2\pi)$$

- Using $\nabla\Psi(\boldsymbol{a}_N^*) = 0$, the term in $\frac{\partial}{\partial\theta_j}\ln p(\boldsymbol{t}_N \mid \theta)$ with explicit dependence on $\boldsymbol{a}_n^*$ is as follows:

$$-\frac{1}{2}\sum_{n=1}^{N}\frac{\partial\ln\left|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right|}{\partial a_n^*}\frac{\partial a_n^*}{\partial\theta_j}$$

- Using $\frac{\partial}{\partial x}\ln|\boldsymbol{A}| = Tr\left(\boldsymbol{A}^{-1}\frac{\partial\boldsymbol{A}}{\partial x}\right)$ and $\boldsymbol{W}_N = diag\left(\sigma(a_i)(1-\sigma(a_i))\right)$, $d\sigma/da_i = \sigma(a_i)(1-\sigma(a_i))$, it becomes

$$-\frac{1}{2}\sum_{n=1}^{N}Tr\left[\left(\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right)^{-1}\underbrace{\frac{\partial\boldsymbol{W}_N}{\partial\sigma(a_n)}}_{(1-2\sigma_n^*)}\underbrace{\frac{\partial\sigma(a_n)}{\partial a_n}}_{\sigma_n^*(1-\sigma_n^*)}\right]\frac{\partial a_n^*}{\partial\theta_j} = -\frac{1}{2}\sum_{n=1}^{N}\left[\left(I + \boldsymbol{C}_N\boldsymbol{W}_N\right)^{-1}\boldsymbol{C}_N\right]_{nn}\underset{\sigma(a_n^*)}{\sigma_n^*}(1-\sigma_n^*)(1-2\sigma_n^*)\frac{\partial a_n^*}{\partial\theta_j}$$

Learning Hyperparameters

- We finally can compute $\partial a_n^* / \partial \theta_j$ as follows : $\boldsymbol{a}_N^* = \boldsymbol{C}_N \left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N \right) \Rightarrow$

$$\frac{\partial \boldsymbol{a}_N^*}{\partial \theta_j} = \frac{\partial \boldsymbol{C}_N}{\partial \theta_j} (\boldsymbol{t}_N - \boldsymbol{\sigma}_N) - \boldsymbol{C}_N \boldsymbol{W}_N \frac{\partial \boldsymbol{a}_N^*}{\partial \theta_j}$$

$$\boldsymbol{W}_N = diag\left(\sigma(a_i)\left(1-\sigma(a_i)\right) \right)$$

$$d\sigma / da_i = \sigma(a_i)(1-\sigma(a_i))$$

- From this we derive:

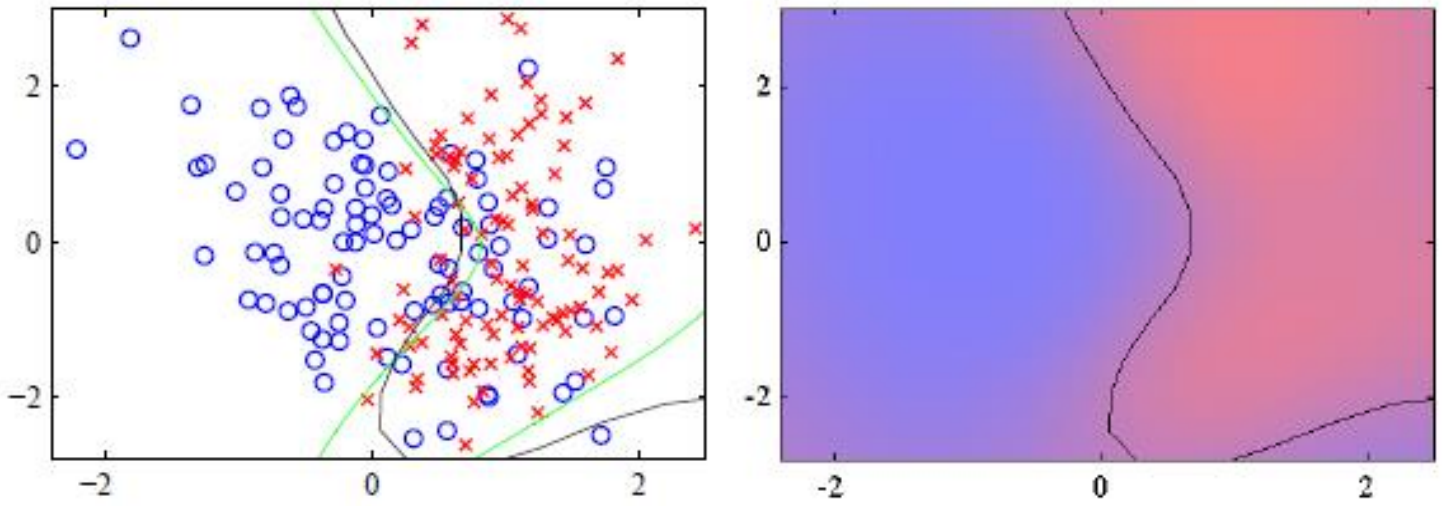
$$\frac{\partial \boldsymbol{a}_N^*}{\partial \theta_j} = \left(\boldsymbol{I} + \boldsymbol{C}_N \boldsymbol{W}_N \right)^{-1} \frac{\partial \boldsymbol{C}_N}{\partial \theta_j} (\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$$

- We now have all the terms in the gradient $\frac{\partial}{\partial \theta_j} \ln p(\boldsymbol{t}_N \mid \theta)$ and can employ gradient optimization techniques.

Binary Classifier Based on Gaussian Process

- We now we have a binary classifier based on a Gaussian process. On left the data together with the optimal decision boundary from the true distribution in green, and the decision boundary from the Gaussian process classifier in black.

- On right the predicted posterior probability for the blue and red classes together with the Gaussian process decision boundary.



Requires installing NetLab and setting the appropriate path

MatLab Code

- Williams, C. K. I. and D. Barber (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**, 1342–1351.

Connections to Neural Networks

- Neural Networks
  - The range of representable functions is governed by the number M of hidden units
  - Within the maximum likelihood framework, they overfit as M comes close to the number of training samples
- Bayesian Neural Networks
  - The prior over $\boldsymbol{w}$ in conjunction with the network function $f(\boldsymbol{x}, \boldsymbol{w})$ produces a prior distribution over functions from $y(\boldsymbol{x})$
  - The distribution of functions will tend to a GP as $M \rightarrow \infty$.
    - In this limit the output variables of the neural network become independent.
    - However, the property that the outputs share hidden units and thus all of them affecting the values of the weights is lost in this limit.

- Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer. Lecture Notes in Statistics 118.

Connections to Neural Networks

- There are explicit forms for the covariance in the case of probit and Gaussian hidden unit activation functions.
  - These kernel functions $k(\boldsymbol{x}, \boldsymbol{x}')$ are nonstationary, i.e. not a function of $\boldsymbol{x} - \boldsymbol{x}'$.
  - This is a consequence of the Gaussian weight prior being centered on zero which breaks translation invariance in weight space.
- By working directly with the covariance function we have implicitly marginalized over the distribution of weights.
- If the weight prior is governed by hyperparameters, then their values will determine the length scales of the distribution over functions.

- Williams, C. K. I. (1998). Computation with infinite neural networks. *Neural Computation* **10**(5), 1203–1216.